DISTRIBUTED FAULT-TOLERANT QUALITY OF SERVICE ROUTING IN HYBRID DIRECTIONAL WIRELESS NETWORKS

THESIS

Larry C. Llewellyn II, Captain, USAF

AFIT/GE/ENG/07-15

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/07-15

DISTRIBUTED FAULT-TOLERANT QUALITY OF SERVICE ROUTING IN HYBRID DIRECTIONAL WIRELESS NETWORKS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Larry C. Llewellyn II, BS

Captain, USAF

March 2007

AFIT/GE/ENG/07-15

DISTRIBUTED FAULT-TOLERANT QUALITY OF SERVICE ROUTING IN HYBRID DIRECTIONAL WIRELESS NETWORKS

Larry C. Llewellyn II, BS

Captain, USAF

Approved:

Kenneth M Hopkinson	2 Mar 07
Dr. Kenneth M. Hopkinson (Chairman)	Date
Scott R. Graham	2 Mar 07
Maj Scott R. Graham, PhD (Member)	Date
[illegible]	2 Mar 07
Maj Paul D. Williams, PhD (Member)	Date

AFIT/GE/ENG/07-15

Abstract

A hybrid mobile ad hoc network (H-MANET) consists of a group of communicating hosts that form an arbitrary network topology by means of any of several wireless communication media – E.g., free space optics (i.e., infrared laser light), or directional radio frequency technology. H-MANET communications represent a diversification in communication technology necessary to solve the stringent end-to-end requirements of the GIG. Of the many challenges in this complex distributed system, the problem of routing based on a predefined set of customer preferences, critical to guaranteeing quality-of-service, is the focus of this research. Specifically, this thesis modifies a cluster based QoS routing algorithm for mobile ad hoc networks with the aim of providing fault-tolerance – a critical feature in providing QoS in the link failure prone environment of mobile networks. Performance of this new fault-tolerant cluster based QoS wireless algorithm is evaluated according to failure recovery time, dropped packets, throughput, and sustained flow bandwidth via simulations involving various node failure scenarios along QoS paths.

Acknowledgments

This product would not be possible without the love and support of my wife. I would also like to express my sincere appreciation to my faculty advisor, Dr. Kenneth Hopkinson, for his guidance and support throughout the course of this thesis effort. The insight, experience, and patience were certainly appreciated.

Larry C. Llewellyn II

Table of Contents

Page

List of Figures

List of Tables

DISTRIBUTED FAULT-TOLERANT QUALITY OF SERVICE ROUTING IN HYBRID DIRECTIONAL WIRELESS NETWORKS

I.Introduction

1.1 Background

This chapter gives a general introduction to the research subject area and an overview of the problem focused on. This section provides motivation for this particular problem and why it is significant. Additionally, in this chapter states the research goal and presents an overview of the remaining document.

Department of Defense Directive 8100.2 (DoDD 8100.2, April 14, 2004) defines the Global Information Grid (GIG) as "…the globally interconnected, end-to-end set of information capabilities, associated processes, and personnel for collecting, processing, storing, disseminating, and managing information on demand to warfighters, policy makers, and support personnel"[11]. Network Operations (NetOps) provides guidance on the essential tasks, Situational Awareness (SA), Command and Control (C2) that the Commander United States Strategic Command (CDRUSSTRATCOM) will use to operate and defend the GIG through the Defense Information Systems Agency (DISA) – a component of United States Strategic Command[5]. A significant function of NetOps is to provide assured net-centric services in support of information superiority – the key enabler to achieve 'Full Spectrum Dominance' as outlined in Joint Vision 2020. Hence, NetOps is CDRUSSTRATCOM's realization of DoDD 8100.2. DISA's vision specifically describes the provided service as global net-centric solutions as follows; "…the provider of global net-centric solutions for the Nation's warfighters and all those who support them in the defense of the nation" [24]. The take-away from DoDD 8100.2, NetOps and the DISA vision is that the GIG is expected to be the globally operating distributed system which will bring this net-centric blueprint to fruition.

'Net-centricity', as described by the DoD, is a robust, globally interconnected network infrastructure with the ability to provide the right information to the right person at the right place at the right time. Similarly, DoDD 8320.2 states that "Net-Centric Warfare (NCW) is an information superiority-enabled concept of operations that generates increased combat power by networking sensors, decision makers, and shooters to achieve shared awareness, increased speed of command, higher tempo of operations, greater lethality, increased survivability, and a degree of self-synchronization" [10]. This forward thinking on the part of civilian and military leadership, a fundamental shift from platform-centric computing to network-centric information resource sharing, has permeated all aspects of DoD leased, owned, and operated information systems. As Alberts and Stein stated, "In essence, net-centric warfare translates information superiority into combat power by effectively linking knowledgeable entities in the battle space"[2].

The synergistic effect of combining net-centric computing with SA, will have a dramatic positive impact on mission effectiveness for all DoD functional areas. Enabling this type of power has far reaching technological implications. That is, to provide the warfighter with the access to relevant data as described by the NCW concept, the associated network will must be dynamic and robust with the ability to reach any location on the globe. Moreover, this network must be able to provide the user with the quality-of-service (QoS) necessary to accomplish the associated mission.

Generally speaking, QoS is a defined level of performance in a data communications network required by a particular type of network traffic. For example, the United States military has recently been field-testing ruggedized versions of common personal digital assistants (PDAs) that have the capability to send and receive tremendous amounts of data. If these devices are to send and receive multimedia traffic, the GIG must provide a certain level of QoS for this multimedia application to be functional. More detailed consideration of QoS and its implications is provided in Chapter Two of this document.

As deployed military units are inherently mobile and generally do not have an existing infrastructure with which to connect, the ad hoc implementation of IEEE 802.11, or similar wireless standard, appears to be well suited for this dynamic application. On the contrary, the bandwidth limitations and poor scalability of this omni-directional communication protocol limit its use in the deployed military environment. Conversely, the combination of free space optic transceivers with high-bandwidth directional broadcast radio frequency technology provide the hybrid communication technologies which can be used to construct the global, dynamic, QoS capable network implicitly described by DoDD 8100.2. Hybrid mobile ad hoc network (H-MANET) communications represent a diversification in communication technology necessary to solve the stringent end-to-end requirements of the GIG. Hence, the H-MANET inherits positive attributes of the MANET while avoiding limiting characteristics attributable to homogeneity and omni-directionality.

Much of this research is a natural extension of the MANET. That is, the models and protocols that exist for the MANET are considered and, when beneficial, are adapted to the H-MANET. Additionally, the assumption is made that each node in the H-MANET has an 802.11 transceiver as well as directional communication interfaces. The purpose for this 802.11 functionality will be evident later. The H-MANET is primarily envisioned as an augmentation to the communication technologies employed at the network edge, providing connectivity between the fixed GIG infrastructure and mobile military units as well as intra/inter connectivity among these mobile military units**.** The H-MANET is therefore not 'ad hoc' in the true sense, but instead a pseudo ad hoc network which shares characteristics of both fixed and ad hoc networks. This research assumes nodes in the H-MANET have multifarious communication interfaces, each having the capability to send and receive data via one of several communication media. Throughout this thesis, when H-MANET is used this is the implied framework. Note that problems related to determining directional antenna positioning are not considered here. It should also be noted that although the H-MANET (depending on the employed communication technology) exhibits many important qualities which make it desirable over standard homogeneous wireless networks (i.e., low power consumption, low probability of interception, etc) the primary characteristic referenced in this work is bandwidth.

1.2 Problem Statement

An illustrative scenario supported by the GIG might be transmission of real-time surveillance video capturing terrorist leadership activity. The purpose of this surveillance data could be preparation for imminent aggressive maneuvers against this enemy target. Imagine that the video transmission is being sent from an embedded covert team to decision makers at DoD headquarters. These two communicating blue forces would also likely require the ability to transmit or receive additional data such as voice and position. Positioning and location update data can be handled well in a best-effort network such as the Internet; however, in order to supply the warfighters in this scenario with the capability to send and receive reliable real-time video and voice packets, the involved network must be able to provide a certain level of QoS.

Consider the network for this application implemented such that the traffic is routed via any available communication technology able to provide the necessary bandwidth and low probability of interception. Further, given the location, traffic requirements, and the potential network dynamics, the optimum involved communication hardware might be a mix of the wireless directional technologies currently employed by the United States Armed Forces. For example, the remote covert team sends its transmission up to a circling cluster of Predator UAVs (far enough away that they are undetectable), which sends the traffic to a forward deployed convoy. The convoy then transmits the signal up to a satellite via an International Marine/Maritime Satellite (INMARSAT) transmitter. Finally, the satellite forwards the data on to DoD headquarters. The path from DoD headquarters back to the covert team follows the same route in reverse. Consequently, this scenario represents an ad hoc hybrid directional wireless network which has the task of supplying QoS in an effort to provide the right information to the right place at the right time.

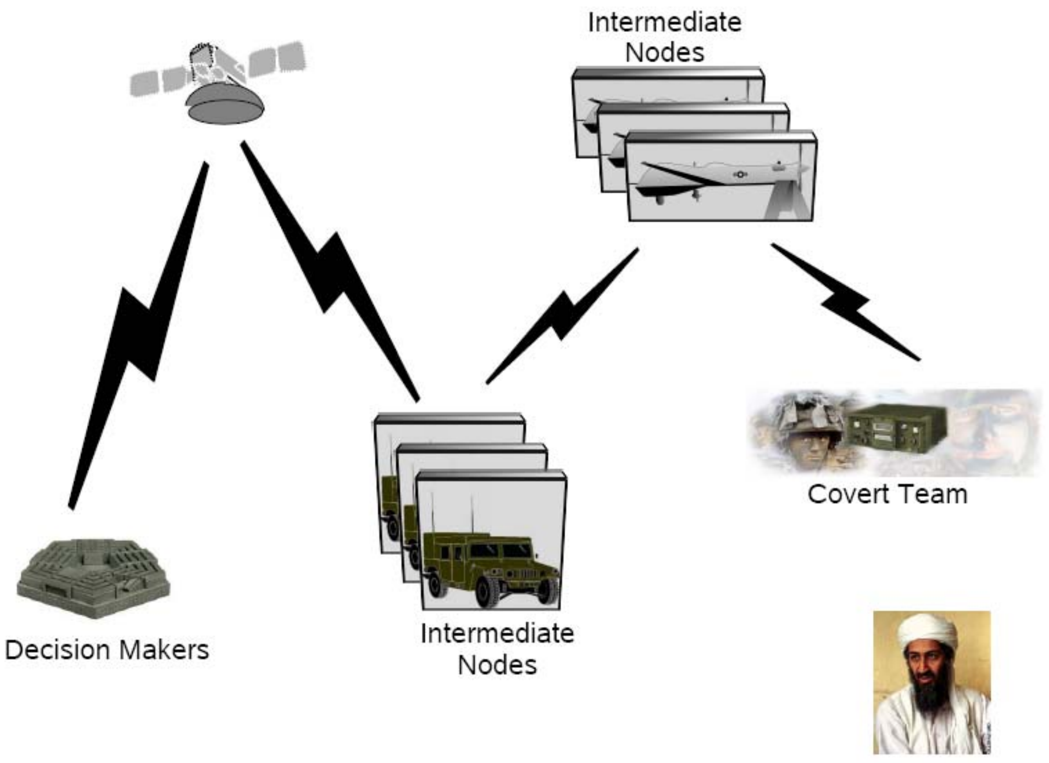


Figure 1.1. Illustrative scenario supported by the GIG

This is but one of countless scenarios which could take advantage of the information resources of the GIG as it has been defined in DoDD 8100.2. This thesis addresses three main inter-related problems associated with the intrinsically complex communication paradigm illustrated here and any like it involving multimedia traffic in a mobile network with intermediate connecting nodes. These three problems are briefly described in the following paragraphs.

The first problem deals with scalability. In the types of ad hoc hybrid directional wireless networks described so far, in order to determine feasible routes, nodes must be aware of what resources are available. With most ad hoc wireless networks that support QoS, each node functions not only as a source or destination but also as a router. In the standard distributed reactive routing implementation, if a node does not know the QoS parameters of its neighbors it broadcasts the route request packet and the neighboring nodes share their QoS parameters using the broadcast packets. Therefore, nodes discover the QoS parameters of their neighbors and negotiate the QoS path via broadcasts. These broadcast packets have the potential to flood the network. A clustered approach to the ad hoc QoS network can lower this unnecessary communication overhead to a more scalable level by limiting inter-cluster routing control communication to gateway nodes only. In this way, only small fraction of network nodes sends routing control packets, thereby significantly reducing communication requirements. It is important to note here that it is assumed all nodes in the H-MANET have 802.11 transceivers, which are used primarily for the purpose of cluster construction and dissemination of available resource information.

The second problem deals with QoS routing directly. Multi-constrained routing is NP-complete[30]. In one heuristic, the QoS routing paradigm can be modeled as a multicommodity flow problem where flow priorities, user preferences and link characteristics are used to compute an optimal set of paths. This process is very similar to multi-protocol label switching technology, except that in this instance, flows are allowed to split, whereas with multi-protocol label switching this is generally not the case. This transformation from multi-constrained routing to the multicommodity flow problem allows the solution to be found in polynomial time. More detail on this methodology is presented in Chapter Two of this document.

The third problem in the described scenario and the primary focus of this research is minimizing QoS impact in the face of network failures. That is, if a supporting intermediate node along the connection from the covert team to DoD headquarters should fail or move, what actions should the network take? More specifically, if the traffic is routed such that it passes through multiple nodes in the convoy (i.e., the data must be routed through several convoy nodes before it can be uplinked to the satellite via the INMARSAT) and one of the supporting convoy nodes fail, how does the network respond? In the worst case, if the network has no built in protocol to respond to this link breakage, the connection will have to be rerouted from the source and QoS will have to be re-established. This global fault-tolerance method means the source will have to re-compute and renegotiate a new QoS path which, depending on the network size, could be costly in terms of computation and communication time necessary for path negotiation (negotiation includes implementation of the new path). Further, if the situation were such that multiple sources were using the failed node in their QoS path, each affected source must re-compute and negotiate a new path. Despite being costly, research suggests that this method of handling failures is commonplace for this category of data networks.

Conversely, if a fault-tolerant algorithm is implemented which makes it possible for the intermediate nodes (i.e., the convoy nodes in this example) to efficiently repair the failed connection locally; it is likely that the associated connections will suffer only a minor disruption, if any. This mitigated disruption time could easily mean the difference between receiving time sensitive, mission critical information, or not. The research in this thesis demonstrates that a solid, local fault-tolerant algorithm has significant benefits over the standard practice of rerouting from the source for QoS traffic.

1.3 Preview

In summary, with an efficient distributed fault-tolerant protocol, QoS disruption time can be significantly mitigated for the QoS supporting mobile ad hoc network. It is proposed that a cluster based scheme holds great promise as the framework for a distributed protocol to address the unique characteristics of the mobile ad hoc network. The focus of this research is to develop an efficient distributed fault-tolerant QoS routing algorithm suitable for the hybrid mobile ad hoc wireless network. Note that this preliminary work is part of a larger effort to enable the robust H-MANET functionality described previously.

This chapter covered a general introduction to the research subject area and an overview of the focus of this thesis. Chapter Two presents introductory material on the QoS problem as it applies to wired networks and builds on this with the topic of QoS in the MANET. The intent is for those with general knowledge in the broad subject matter to be able to understand the particular area of this research. Chapter Two also contains an overview of literature that supports key design decisions in development of the distributed fault-tolerant cluster based QoS protocol. Chapter Three presents the developed fault-tolerant protocol. Chapter Four documents the experimental methodology used in this research and presents the performance analysis of the competing algorithms. Chapter Five concludes the document with a summary of results and recommendations for further research and development.

II.Literature Review

2.1 Introduction

This chapter presents a review of introductory material and related literature. First, a discussion of the general quality-of-service (QoS) routing problem is presented as it applies to wired networks. This involves the presentation of a recent work exploring scalability in MPLS-based networks. The general QoS routing discussion then leads into two heuristic algorithms with the primary goal of minimizing memory and runtime complexity for the multiconstrained QoS routing problem.

This thesis focuses on fault-tolerance in the QoS supporting H-MANETs. Since nodes in an H-MANET have the ability to send and receive omni-directional wireless traffic, we investigate the protocols and communication models associated with the MANET and applicable to fault-tolerance and QoS support. The remainder of the chapter includes the following topics: 1) the MANET, its applications and the general ad hoc routing problem, 2) the QoS MANET model and associated QoS routing problem, 3) a fully distributed cluster based QoS routing method with a discussion of proposed modifications to enable improved QoS routing and fault-tolerance. Finally, although little literature exists on the topic, two techniques applied to solve the fault-tolerant problem in the QoS supporting MANET are covered– the first involving a repair method comparable to the clustered protocol developed for this work and the second concerning a more network resource-demanding approach termed *multi-level path redundancy*.

2.2 Quality-of-Service Routing

The Internet in general is based on a best-effort framework in which network traffic is sent and received without any guarantees on reliability, latency, or other quality metrics. Despite that fundamental premise, the Internet is used today for various real-time services, such as videoconferencing and voice-over-IP (VoIP). These applications require a certain quality of service if they are to be fully functional. QoS is described by RFC2386 as a set of service requirements (e.g., bandwidth, delay, probability of packet loss, variation in delay or jitter, and so on) to be met by the network while transporting a flow associated with a particular application. This implies a guarantee by the network to satisfy a predetermined service performance constraint for the user in terms of any combination of the previously mentioned network parameters (or others, as this is not an exhaustive list).

The network can be modeled as a graph (*N*, *E*). *N* nodes of the graph represent switches, routers, and hosts. *E* edges represent communication links which are undirected and not necessarily symmetric; however, the example shown in Figure 2.1 assumes symmetric edges. Every edge has a characteristic measured by the associated QoS metric (bandwidth, delay, etc) represented in the network. Further, every node has state, which should be considered when determining a feasible path. For example, the true link bandwidth is the minimum of the link bandwidth and the maximum rate the node can put data on to the link. For the purposes of this study, the node states that affect the related link states are considered negligible. Also, note that this research is primarily concerned with unicast routing.

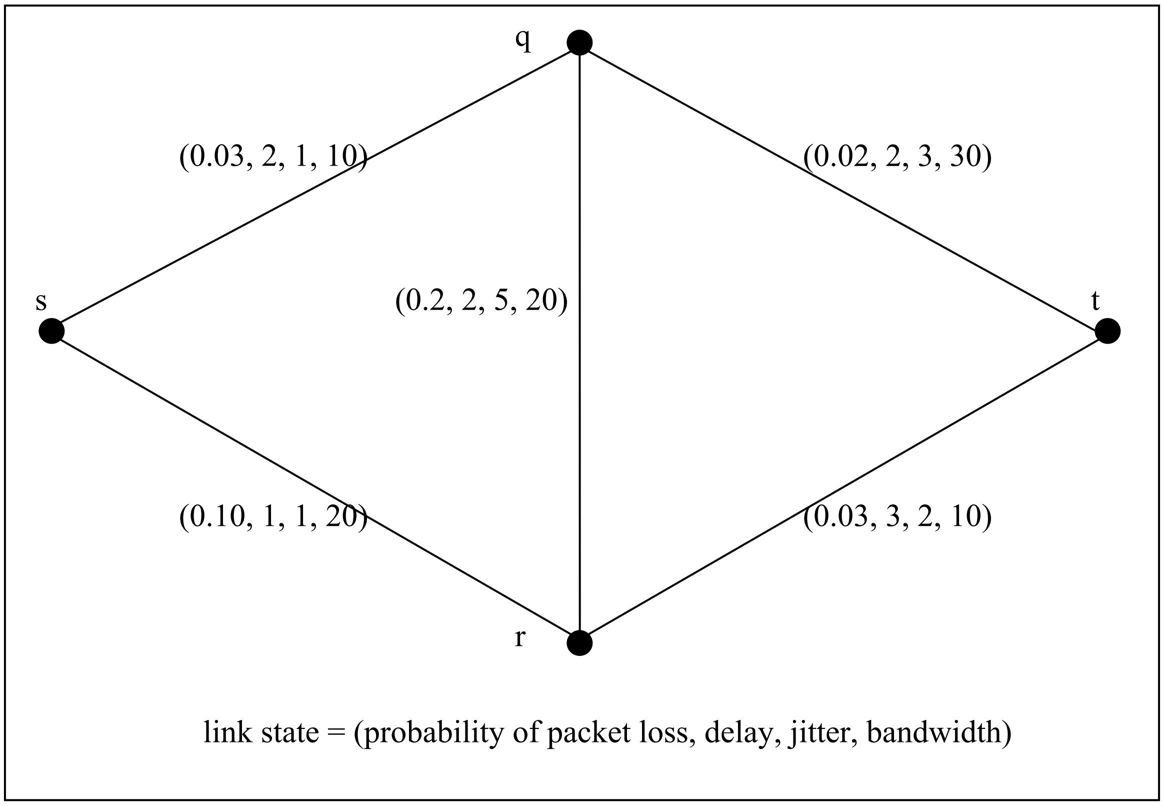


Figure 2.1. QoS network illustrating link state

QoS routing requests are specified in terms of constraints. The multiconstrained path QoS routing problem involves finding routes that satisfy multiple independent QoS constraints. Stated more formally, given a directed graph $G(N, E)$, a source node *src*, a destination *dst*, $k \geq 2$ weight functions $w_1 : E \rightarrow R^+$, $w_2 : E \rightarrow R^+$, …, $w_k : E \rightarrow R^+$, k constants $c_1, c_2, \ldots, c_k$; the problem is to find a path p from *src* to *dst* such that $w_l(p) \leq c_1$, …, $w_k(p) \leq c_k$ [30] (where k is the number of constraints and $1 \leq l \leq k$). An example of 2-constrained routing is delay-packet loss constrained routing or finding a route with bounded end-to-end delay and bounded end-to-end packet loss probability. After finding the feasible route, the second part of the QoS routing process is the task of reserving the required resources along this newly discovered route. That is, the required bandwidth, queues, or other associated network resources needed to provide the required QoS must be set aside for use by the intended source. Once the transmission is finished, these resources are released. Additionally, the knowledge that these resources are in use must be tracked so that any potential user will have an accurate account of the available resources. When the term QoS is used in this document it denotes calculating the route and reserving the associated resources along the calculated route.

The multiconstrained QoS routing problem is difficult because different constraints can conflict with one another. For example, a route may provide the required end-to-end delay and not be able to provide the necessary packet loss probability end-to-end or vice versa. Multiconstrained path QoS routing is known to be NP-hard [25]. There are three general categories of QoS routing for wired networks: centralized routing, distributed routing and hierarchical routing.

2.2.1 Quality-of-Service routing using centralized algorithms.

Centralized algorithms employ a method known as source routing. This scheme requires that the source maintain complete global state knowledge. Consequently, the route computation is centralized at the source. A link state protocol is used to periodically update the global state at all nodes in the network. Some of the advantages to source routing are that it avoids the problems of distributed algorithms such as distributed state snapshot and deadlock detection; however, some major drawbacks are scalability and the requirement of an accurate account of the network's state. The global network state at every node has to be updated frequently enough to cope with the dynamics of the network. This has the potential to make communication overhead excessive for large networks. As the network grows the problem of maintaining an accurate global network state becomes even more impractical. In addition, with network growth comes increased overhead necessary to distribute this information and increased routing algorithm run time. Furthermore, the memory requirement for each node becomes large as the network grows. Even with the capacities available on today's networks, this dissemination will take time, and during that time the state of the network will undoubtedly change. The end result is that routing decisions will potentially be made based upon stale information. Additionally, even if the network state information is accurately received, it is possible that the network topology will change during the route calculation process. Lastly, in terms of computation time, since all route calculations are performed at the source, the routing algorithm used in this centralized scheme must factor in this resource limitation and therefore minimize computational complexity.

2.2.2 Quality-of-Service routing using distributed routing algorithms.

With distributed routing, algorithms can be made more scalable as the path computation is inherently distributed among the intermediate nodes between the source and the destination. Many existing distributed algorithms execute routing decisions on a hop-by-hop basis, but rely on global network state information (like that necessary for centralized QoS routing algorithms). As a result, these algorithms suffer scalability problems similar to source routing algorithms. Distributed algorithms, which do not need global state information and use only local state data, exist; however, they tend to send more overhead messages since their view of the network is incomplete. These algorithms also suffer potential loops as network views may differ between nodes; however, loops can fairly easily detected when the routing message is received by a node for the second time. It is also difficult to design efficient distributed heuristics for the multiconstrained routing problem in the absence of detailed topology and link state information.

2.2.3 Quality-of-Service routing using hierarchical routing algorithms.

Another routing methodology, hierarchical routing, exists which shares some of the advantages of both centralized and distributed routing. Hierarchical routing has been employed in the past to solve the scalability problems noted with centralized routing. Improved scalability arises from the fact that each node only maintains a partial global state such that groups of nodes are aggregated into logical nodes. The effect of this method is to create a logical network that is logarithmic in size compared to the actual network. With this approach, a source routing algorithm can be employed at each hierarchical level to find feasible paths based on the aggregated states maintained at each node. The disadvantage is that with this aggregated view of the network, the possibility of inaccuracy becomes a problem since it is possible that a logical node (aggregated group of nodes) may be a large complex network that is incorrectly represented. The fundamental dilemma here is that aggregation of large complex networks into a logical node is a problem that has yet to be accurately solved [8].

2.2.4 Flows and Flow Networks.

A flow network is defined by the National Institute of Standards and Technology as "A weighted, directed graph with two specially marked nodes, the source *s* and the sink *t*, and a capacity function that maps edges to positive real numbers, $c : \mathrm{E} \rightarrow R^{+}$" [19]. A flow network is modeled here as a connected, directed graph $G = (V, E)$ (where V is the set of all nodes and E is the set of all edges) such that each edge $(u, v) \in E$ has a capacity $c(u, v) \geq 0$. Further, in this directed graph, there are two special nodes – source *s* with only outflows and sink *t* with only inflows. A flow can then be defined as a real-valued function f: $V \times V \rightarrow R$ that assigns flow values to the edges of a flow network, $f(u, v)$. Moreover, this function must satisfy the following properties [9]:

Flow conservation: For all nodes in the network, other than the source and destination, the incoming flow must be the same as the outgoing flow:

$$\forall u \in V - \{s,t\}, \sum_{u \in V} f(u,v) = 0 \text{ [9]}$$

Capacity constraint: For all nodes in the network, the flow between two nodes must be less than, or equal to, the capacity of the connecting edge.

$$\forall u \in V, \ f(u,v) \leq c(u,v) \text{ [9]}$$

Skew Symmetry: For all nodes in the network, starting from either node of every edge, the flow is the same amount, but reversed direction (i.e. for an edge *e* connecting vertices *u* and *v*, the flow from *u* to *v*, $f(u, v)$, equals the negation of that flow in the opposite direction $-f(v, u)$.

$$\forall (u,v) \in V, f(u,v) = -f(v,u) \text{ [9]}$$

2.2.5 Maximum flow and Multicommodity flow Problems.

The maximum flow problem can be described as the task of finding the path in a flow network from a source to a destination which yields the flow of maximum value [9]. The maximum flow problem can be considered a special case of the more complex multi-commodity flow problem with a single commodity, namely the path capacity. Several algorithms exist which solve the maximum flow problem in polynomial time. One such algorithm is Edmunds-Karp, a specialization of the Ford-Fulkerson, which finds paths using a breadth first search and has a runtime complexity of $O(VE^2)$ [9] .

As an illustration of the multicommodity flow problem, suppose that a manufacturing company builds widgets at their manufacturing plant in Dayton, OH and builds doodads at a facility in Fairfax, VA. Further, the company has a storage facility for the widgets located in Charlotte, NC and another storage facility for doodads in Washington, DC. Each item (widget and doodad) must be shipped each day from the factory to the associated storage facility. The capacity of the shipping network (The Interstate System) is constant, and the different items (or commodities) must share the same network. In this problem, again there is a directed graph in which each edge has a non-negative capacity. At the heart of this problem is the idea that there are r different commodities, K_1, K_2, K_3, …, K_r where commodity i is specified by $K_i = (s_i, t_i, d_i)$ such that s_i is the source of the commodity, t_i is the commodity sink, and d_i is the desired flow value for commodity i from s_i to t_i. A flow for commodity i, denoted by f_i, (where $f_i(u_i, v_i)$ is the flow of commodity i from node u_i to node v_i) is defined here to be a real-valued function that satisfies the node conservation constraints. Then the multicommodity flow problem is to find flows from a source to a destination that satisfy the flow properties previously described (flow conservation, skew symmetry, and capacity) and meet some objective function criteria so that the sum of flows on any edge does not exceed the capacity of the edge. For the maximum multicommodity flow problem, the objective is to maximize the sum of the flows: max $\sum |f_i|$ [13].

2.2.6 Quality-of-Service routing using MPLS and Multicommodity Flows.

Multi-Protocol Label Switching (MPLS) is an advanced forwarding QoS supporting scheme in which incoming packets are first assigned a label by a label edge router (LER). Label switched routers then use these labels to forward packets along a label switched path. The label switched paths are designed by network engineers for a variety of purposes ranging from guaranteeing a class of service to routing around congested links. MPLS consists of a layer between the network layer and the data link layer. MPLS, as described by RFC3031, generally uses multiconstrained routing in determining dynamic routes. The most popular construction of the QoS routing problem that implements multicommodity flows is based on the MPLS architecture in which a multicommodity flow optimization is performed where an objective function is minimized with respect to the types of flow subject to multicommodity flow constraints [20]. In essence, MPLS provides the mechanism for reserving a single path, while multicommodity flow optimization finds an optimal set of paths given multiple sets of commodities. Multicommodity flow algorithms seek to minimize the objective function, e.g., number of hops, and will therefore choose routes along the shortest path. This optimal solution may split the flow along multiple paths of the network. Most MPLS routing protocols disallow flow splitting, and therefore may produce a less than optimal solution, if any. Multicommodity flow algorithms inherently split flow because the continuous solution is solvable in polynomial time whereas an integer solution, which would not split flows, is not. MPLS and other reservation schemes do not split flows, thereby avoiding increased router complexity (i.e., routers would have to route fractional flows).

Mitra and Ramakrishnan [21] propose a 'scalable' technique which uses the multicommodity flow method for handling the QoS routing problem in a MPLS supported IP network. With this technique emphasis is placed on scalability, which they maintain is the reason they use the multicommodity flow approach (opposed to multiconstrained routing). Conversely, Applegate and Thorup [3] state that implementing the optimal multicommodity flow solution using the technique shown by Mitra and Ramakrishnan [21] has the potential to lead to exceedingly large routing tables, indeed, the worst case would be $\Theta(\mathit{Destinations} \times \mathit{Edges})$ entries. Applegate and Thorup [3] provide a supposed solution to this exorbitant memory requirement; however, their solution still requires an algorithm similar to Mitra and Ramakrishnan [21] as a "front-end" that produces an optimal path (or set of optimal paths) which Applegate and Thorup then use as input to their algorithm. Since Mitra and Ramakrishnan's algorithm may require routing tables as large as $\Theta(\mathit{Destinations} \times \mathit{Edges})$ entries, Applegate and Thorup's claim is only that "…given a good solution, we can implement with $K \times (\mathit{Destinations} + \mathit{Edges})$ table entries" (where K is the number of traffic classes)[3]. Hence, the memory implications involved with both solutions are prohibitive for the

MANET application and the fundamental question of whether or not the MPLS approach can scale for larger networks remains.

2.2.7 Quality-of-Service routing using heuristic algorithms.

Xin Yuan [30] proposes two heuristic algorithms for the multiconstrained QoS routing problem which deal with $k \geq 2$ constraints. These competing heuristics are applied to the extended Bellman-Ford algorithm (EBFA), which is a variation of the Constrained Bellman-Ford algorithm developed by Ron Widyono [29] to solve k-constrained QoS routing problems (where k is a small constant). For completeness, it is noted here that the original Bellman-Ford algorithm computes the shortest path in a weighted directed graph (where some of the edge weights may be negative). The EBFA computes the feasible path given multiple constraints. The following definitions are necessary to understand the functionality of EBFA shown in Figure 2.2:

EBFA Definitions

- *PATH(u)* – all optimal QoS paths found so far from the source *src* to *u* [30]
- $e = u \rightarrow v$ – represents connecting edge e from node u to node v which has k independent weights [30]
- $w(u \rightarrow v)$ – the k independent weights ($w_1(e)$, $w_2(e)$, …, $w_k(e)$) used to represent the weights of a link from u to v [30]
- c – the constraint vector for a given request (i.e., $c = w_1, w_2, \ldots, w_k$) [30]
- for path $p = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ (where $v_k \in V$), $w_l(p) = \sum_{i=1}^{k} w_l(v_{i-1} \rightarrow v_i)$

  and $1 \leq l \leq k$

- $w(p) \leq w(q)$ denotes $w_l(p) \leq w_l(q)$ for all $1 \leq l \leq k$ – therefore $w(p)$ is the vector sum of all constraints of all edges in path p [30]

```
(1)   $Foreach\ w(p)\ in\ PATH(u)$
(2)       $flag = 1$
(3)       $Foreach\ w(q)\ in\ PATH(v)$
(4)           $if\ (w(p)+w(u,v) \geq w(q))\ then\ flag = 0$
(5)           $if\ (w(p)+w(u,v) < w(p)+w(u,v)\ to\ PATH(v)$
(6)       $if\ (flag = 1)\ then\ add\ w(p)+w(u,v)\ to\ PATH(v)$

$BELLMAN-FORD(G, w, c, src, dst)$
(1)   $For\ i = 0\ to\ |V(G)|-1$
(2)       $PATH(i) = \phi$
(3)   $PATH(src) = \{0\}$
(4)   $For\ i = 1\ to\ |V(G)|-1$
(5)       $Foreach\ edge\ (u,v) \in E(G)$
(6)           $RELAX(u, v, w)$
(7)   $Foreach\ w(p)\ in\ PATH(dst)$
(8)       $if\ (w(p) < c)\ then\ return\ "yes"$
(9)   $return\ "no"$
```

Figure 2.2. EBFA for the multiconstrained QoS routing problem [30]

Lines 1 through 3 of the BELLMAN-FORD() procedure initialize the algorithms variables. Lines 4 thorough 6 of the BELLMAN-FORD() procedure call the RELAX() routine $|V| \times |E|$ times. Once the RELAX() procedure returns, all optimal QoS paths from node *src* to *dst* are stored in the set *PATH*(*dst*). Lines 7 and 8 check whether there exists an optimal path that satisfies the QoS constraint.

On the RELAX() procedure, for the weight vector $w(p)$ of each path p in *PATH*(u) from *src* to u and for the weight vector $w(q)$ of each path q in *PATH*(v) from *src* to destination *v*, if the current path is better or as good as the new path, keep the current path; otherwise put the new path in place of the current path. In terms of runtime/memory complexity the time and memory needed to execute the RELAX() procedure is dependent upon the number of optimal paths in the sets *PATH*(*u*) and *PATH*(*v*). Since the number of optimal QoS paths from *src* to *u* and from *src* to *v* can be exponential with respect to the size of *V* and *E*, the runtime/memory requirement of EBFA may also grow exponentially [30].

The motivation of both Yuan heuristics is to limit the number of optimal QoS paths maintained in each node and thereby lower the runtime/memory complexity of the heuristics. It should be mentioned that Xin Yuan's paper considers centralized algorithms and assumes that the global network state information is accurate. The first heuristic, the limited granularity heuristic, is guaranteed to obtain approximate solutions in polynomial time. The general concept is to first reduce the NP-complete problem to a simpler one which can be solved in polynomial time. This is done by making use of finite domains, such as bounded ranges of integer numbers, to approximate the infinite number of values that can be used to represent QoS parameters. The proof that this method provides a correct solution for the $k = 2$ case is shown by Chen Shigang [27]. It should be noted that in Xin Yuan's paper, the application is the general *k*-constrained routing problem instead of the 2-constrained problem. The time complexity of the limited granularity heuristic is claimed to be $O(|N|^k|E|)$ for an *N* nodes and *E* edges network with *k* independent QoS constraints. Given the time complexity of the limited granularity heuristic, it is no surprise that performance analysis of this heuristic shows it is inefficient when $k > 3$ (algorithm runtime increases significantly with very small increases in k).

Yuan's second heuristic is the limited path heuristic. In general, the method attempted with this algorithm is to focus on the cases that occur most frequently and solve these cases efficiently and effectively – this done by limiting the number of optimal QoS paths in each node. The difficulty is in deciding what the value of the maximum number of optimal QoS paths in each node should be. The paper goes on to show that if the number of QoS paths in each node is $O(|N|^2\lg(|N|))$ (where N is the set of nodes representing routers) the probability that all optimal QoS paths are recorded by the heuristic is high and hence the probability of finding the QoS path that satisfies the required constraints when such a path exists is high. The performance of the limited path heuristic shows that it requires much less resources (memory and computation time) then the limited path heuristic. It should also be noted that the limited granularity heuristic guarantees finding an approximate solution, while the limited path heuristic cannot provide such a guarantee. The disadvantage of both heuristics is that they employ source routing which, as mentioned previously, has various negative scalability implications.

2.3 Mobile ad hoc networking

In a mobile ad hoc network (MANET), hosts have no network infrastructure with which to connect. The MANET embodies a complex distributed system in which wireless mobile nodes can freely and dynamically organize into a temporary "ad-hoc" network topology, thereby allowing devices to build an internetwork in areas with no pre-existing communication infrastructure. Further, nodes have the ability to randomly move, leave or join the network. Hence, there is potential for the network topology to change often and in unpredictable ways. The applications of this technology involve, but are not limited to conferences, classrooms, disaster-recovery and military tactical operations. The advent of ad hoc networks is certainly not recent; however, the proliferation of wireless technologies such as 802.11 and Bluetooth® has brought this networking model to the forefront of many research projects. In particular, multi-hop routing, one of the most important protocols in the MANET – enabling communication between nodes that are not directly connected – has experienced a significant amount of development. Most of the protocols established for multi-hop routing set up and maintain best-effort routes. In the following section, a short survey of these protocols is presented.

2.4 Routing in the MANET

In wired networks, significant topology changes are uncommon as the associated nodes are largely static. Further, a specific subset of network nodes is provided for the primary purpose of routing packets. On the contrary, in the MANET topology is expected to change and all network nodes cooperate to provide routing services. The characteristics of the MANET (i.e., dynamic topology, bandwidth-constrained, variable capacity links, energy-constrained, scalability, etc) require a fundamental change in routing protocol design. In this distributed unreliable environment, the available routing protocols can be categorized into three design paradigms – reactive on-demand routing, proactive table-driven routing and a proactive-reactive hybrid.

2.4.1 Proactive Routing.

Proactive or table-driven protocols establish routes in advance. Each node must maintain one or more tables to store the routing information. Periodic updates propagate through out the network to account for link changes in the topology. The advantage of this particular MANET routing scheme is that packets can be forwarded with very low latency since the paths are known a priori. This has clear benefits for real-time traffic. The disadvantages are that potentially large amounts of bandwidth can be consumed by the continuous link updates. Further, since much of the stored routing information will not be used, memory management is inefficient in this scheme. To add to the challenge, memory requirements for the QoS routing problem will clearly be larger since more information is required for each stored path (i.e., link state information for all concerned QoS metrics for each edge in the network). These disadvantages once again yield a scalability problem similar to that noted earlier for centralized QoS routing.

Fisheye state routing (FSR) [16] is an example of a proactive routing protocol. FSR is a link state algorithm meaning each node (router) in the network builds a picture of the entire network in its routing table while determining network link state costs. The exchange of information required to determine these network link state costs is expensive. FSR handles this by updating network information for nearby nodes (within the fisheye) more frequently than for nodes more remote. Therefore, given an arbitrary node *i* in the MANET, distance and path quality information will be more accurate for *i* within a certain range (number of hops). As the distance from *i* increases, link state accuracy *i* has of the network decreases. Using the fisheye analogy, the eye of a fish captures with high detail the pixels near the focal point. The detail decreases as the distance from the focal point increases. Due to this distance dependent link state accuracy, FSR is an implicitly hierarchical routing protocol. The tradeoff is improved scalability when compared with typical link state protocols.

Figure 2.3 illustrates the fisheye protocol in a MANET. The different colored circles define the different sets of nodes, or *scopes*, the center node (node 11) can reach within a given number of hops. In this case, three scopes are illustrated – yellow is one hop, blue is two hops and white is any number of hops greater than two. So with this example, nodes in scope one will receive more frequent link state updates than those in scope two and nodes in scope two will receive more frequent updates than those in scope three. The result is a significant reduction in message size since a large fraction of link state entries are not sent in a typical update. Imprecise knowledge of the best path to a distant destination is compensated by the fact that the route becomes progressively more accurate as the packet gets closer to the destination. Note that the number of levels and the radius of each scope will depend upon the size of the network. Since FSR maintains a routing entry for each destination, the protocol avoids the route discovery problem of on-demand protocols and can therefore maintain low packet transmission latency.

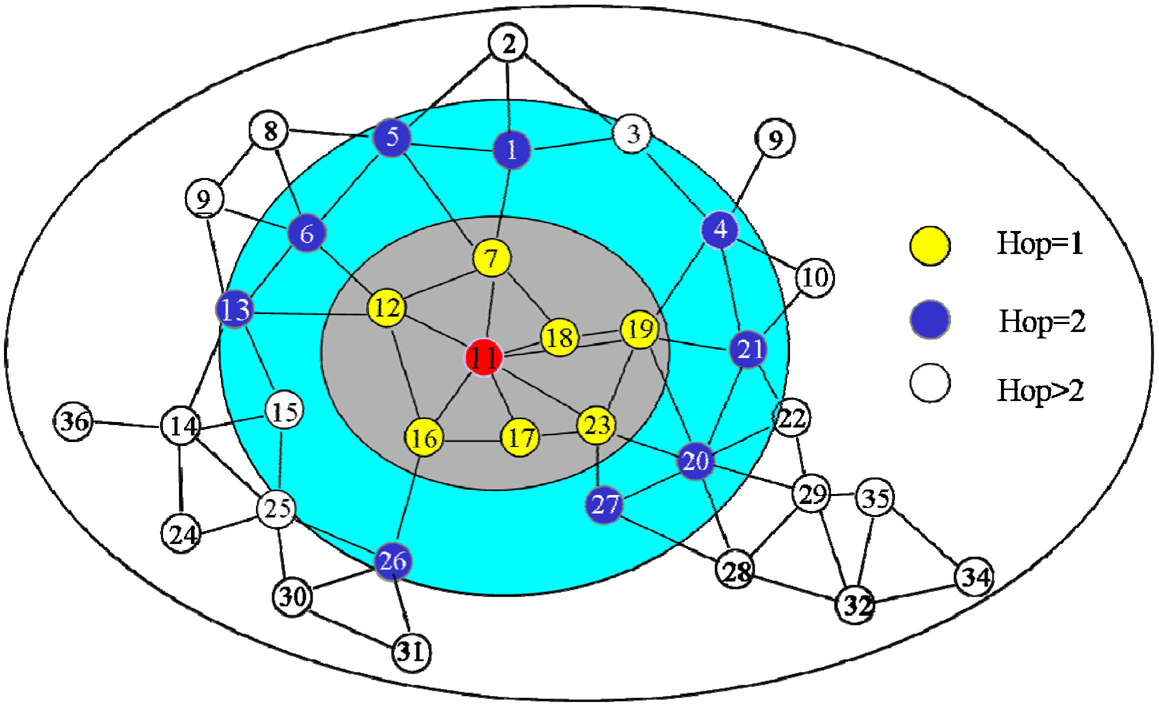


Figure 2.3. Fisheye example [16]

2.4.2 Reactive Routing.

Reactive routing protocols determine routes when requested or on-demand. That is, when a packet needs to be forwarded a route discovery process is initiated by which the node floods the network with route-request packets. Before the route request packet is forwarded, the host appends its address to the route record of the route request packet. When a node with a route to the destination is reached, a route reply packet is sent back. In the worst case, the packet is forwarded all the way to the destination. If links are bi-directional, the route reply will traverse the same path in the opposite direction back to the source. If the links are not bi-directional (i.e., one of the nodes along the discovered path cannot communicate with a preceding node perhaps due to a physical obstruction, power limitations, etc.) the path from the destination to the source will also have to be discovered via flooding. The advantage of reactive routing protocols is that bandwidth (and power) is conserved since constant periodic updates are not required. The disadvantage is delayed packet forwarding since routes are discovered as needed. Also, since reactive protocols are distributed, this drawback is the same as that noted previously for wired distributed QoS routing. Additionally, the flooding process used during route discovery has the potential to strain the available network bandwidth if the volume of route requests is high.

An example of a reactive routing protocol is dynamic source routing (DSR) [4]. DSR assumes all hosts are willing to forward packets for all other nodes. DSR also assumes a small network diameter (i.e., the average minimum number of hops between any two nodes is small). If the path to the destination is known, the sender first constructs a 'source route' in the packet header. This source route contains the address of each host in the network through which the packet is forwarded to reach the destination. Each node in the network maintains a route cache containing the source routes that it knows about. Nodes along the path monitor the operation of the route and inform the sender of any routing errors. This cache is updated as the node learns of new routes. If the path is not known, the route discovery procedure discussed above is executed.

As average node mobility of the network increases, reactive routing produces better performance than proactive routing; however, as the number of connection requests increase proactive routing outperforms reactive routing. The aim of hybrid routing is to strike a balance between these two opposing routing models. With hybrid routing, paths to nodes within a specific distance are proactively maintained. Routes to destinations outside of this area are reactively obtained. The advantages of the hybrid routing design are improved scalability over pure proactive routing and less delay in route discovery for nearby nodes compared to reactive routing.

2.4.3 Hybrid Routing.

An example of hybrid routing is the zone routing protocol (ZRP) [23]. As the name suggests, ZRP divides the network into different zones. Each node in the network has a local neighborhood whose size is defined by a radius length in number of hops. A node proactively maintains routes to all other nodes in its neighborhood. As an example, in Figure 2.4 node *S* is the central node, which has a neighborhood radius of two hops. In this example *L* is outside the neighborhood zone while nodes *A* through *F* are all neighbors (in the zone). *G* through *K* are peripheral nodes or nodes at the extreme end of the zone; these nodes are the most distant destinations in the zone which *S* proactively maintains routes to. It's important to note that with ZRP each node may choose a different zone radius based on its signal strength, transmission power, mobility and so on.

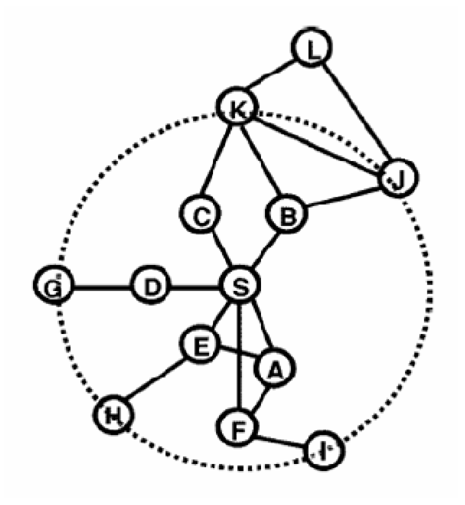


Figure 2.4. Routing zone of radius two hops [23]

ZRP uses the Intrazone Routing Protocol (IARP) to proactively find routes within the routing zone and Interzone Routing Protocol (IERP) combined with Bordercast Resolution Protocol (BRP) to reactively find routes beyond the routing zone. In general, BRP is used to direct route requests initiated by the IERP to the peripheral nodes. BRP provides this packet delivery service by using the map generated by the local IARP.

Consider an illustration of IERP using the network shown Figure 2.5. We begin with node *S* preparing to send data to node *D*. Node *S* first checks to see if node *D* is in its routing zone. Since it is not, node *S* sends a route request query to its peripheral nodes *G*, *H*, and *C*. As nodes *G*, *H*, and *C* do not have a route to node D, they forward the route request to their peripheral nodes specifically B,F,E, and A. Since node *B* has a route to *D*, node *B* responds to *H* with the forwarding path *S*-*H*-*B*-*D*.

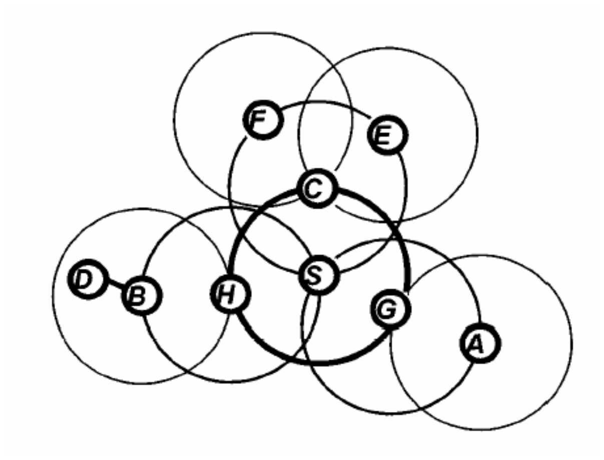


Figure 2.5. IERP example with a zone radius of one [23]

This method of sending route requests to peripheral nodes intuitively seems to be an efficient method of on-demand route discovery; however, flooding is a very real problem. Since routing zones overlap it is possible that a route request will be forwarded to all nodes in the network. To combat this problem ZRP implements a query detection process by which nodes eavesdrop allowing more nodes to learn what queries have already been forwarded. In this scenario, if a previously forwarded request is seen it can be terminated. If this problem is not dealt with correctly, the overhead generated can potentially be worse than that of flooding based queries [23].

The MANET routing protocol designs discussed so far are designed for best-effort level network service; therefore, in their current state they are not adequate for applications with QoS requirements such as multimedia audio and video. For this type of network traffic a proactive-like design makes the most sense since with proactive protocols routes are always available. This allows the QoS route calculations to be executed with lower latency than the reactive scheme. The problem encountered with a proactive solution is scalability. Hybrid protocols share a portion of the advantage of proactive protocols since they proactively maintain a percentage of routes; however, they also share the disadvantage of on-demand reactive routing since destinations outside the routing zone must be found using a route discovery process. Additionally, careful measures must be taken to ensure route request packets do not create flooding broadcast problems. Table 2.1 illustrates the tradeoffs each of these routing paradigms involves. Of the many challenges that exist in the complex MANET system, the problem of QoS routing and route maintenance (or fault-tolerance) is the focus of the following paragraphs.

Table 2.1. Routing Protocol Tradeoffs

Attribute	Proactive	Reactive	Hybrid
Route availability	**Always available**	**On demand**	**Dependent upon location of destination**
Volume of control traffic	**High**	**Low compared to Proactive**	**Lowest**
Periodic updates Required	**Yes**	**No**	**Yes, within a certain area**
Delay	**Low**	**High**	**Dependent upon location of destination**

2.5 Quality-of-Service Routing and Ad Hoc Networks

The QoS model for the MANET can again be modeled as a graph (N, E) of N nodes and E edges. All nodes in this case serve as routers and hosts. Edges E represent communication links which are directed and symmetric. Moreover, E and V change over time depending on node mobility. It is also assumed that each node transmits a beacon packet at periodic intervals. This beacon packet contains the node's unique node identification. The QoS model for the MANET is similar to that previously presented for the wired network in that every edge has a characteristic measured by the associated QoS metric (bandwidth, delay, etc) represented in the network. The same tasks for implementing QoS in the wired domain (i.e., calculating the feasible path, reserving path resources, tracking resource reservations, and so on) are in effect for QoS in the MANET. Perhaps not as obvious is the effect the mobile ad hoc environment has on implementing QoS.

First, the centralized wired QoS routing algorithms require accurate global state information for efficient execution. This is clearly not a reasonable requirement given the dynamic nature of the MANET and the time required for updates to propagate through the network.

Second, when updating the necessary nodes with topology changes, care must be taken to avoid interfering with currently supported QoS connections. This is a potential problem caused by the limited resources (bandwidth), omni-directional nature(e.g., hidden terminal problem) and dynamic nature of the MANET.

Third, and of primary interest to this thesis, the occurrence of QoS connection-link breakage due to node mobility is a very real and likely event; however, recent work found in the literature scarcely addresses this problem. These consequences of the MANET suggest that support for fault-tolerant QoS in this environment is an intractable problem; however, recent research shows there are logical paths to fault-tolerant QoS realization even in this unpredictable domain. Since current wired network QoS routing algorithms are unsuited for the MANET, a new protocol designed for these types of network conditions must be implemented.

It is noted here that high churn, or excessive node mobility, can cause QoS requirements to become unreachable. Excessive node mobility implies that the topology changes before the network updates can propagate to all the relevant nodes[7]. Chakrabarti and Mishra call this characteristic *combinatorial stability*. "An ad hoc network is *combinatorially stable* if and only if the topology changes occur sufficiently slowly to allow successful propagation of all topology updates as necessary" [7]. For the remainder of this thesis, only *combinatorially stable* networks are considered.

2.6 Quality-of-Service routing using fully distributed cluster based routing

Nargunam and Sebastian [22] present a fully distributed cluster based (FDCB) routing algorithm aimed at QoS routing in mobile ad hoc networks. With FDCB routing the issues of scalability encountered with centralized routing are circumvented. This is because the FDCB method inherits the properties similar to that of hierarchical routing in that each node in a cluster only has to maintain QoS information for the other cluster members which is a fractional portion of the entire network. Therefore, an increase in the number of nodes in a network should not demand significant increases in memory or algorithm runtime. Further, since there is no requirement for global network state to be shared and maintained by all, there is no concern for the information dissemination overhead incurred by the centralized routing algorithms. The FDCB routing process is as follows. If the source and destination of a flow are not located in the same cluster, the source sends a route request packet to the gateway node which then forwards the packet to the adjacent cluster(s). As long as the receiving intermediate gateway nodes and links can support the requested QoS constraints, this process is repeated until the destination is located. The discovered path is then sent back to the source at which point the required resource reservations are made and the traffic is sent. The distributed properties of the FDCB routing algorithm allow it to avoid the problem of unmanageable shared global network state information. FDCB's distributed routing scheme means packets will suffer initial transmission latency due to the time required for route discovery. Additionally, route requests may not flood the network in the traditional sense since a clustered architecture is used; however, precautions must be taken to ensure route queries propagate from the source to the destination as efficiently as possible and eventually terminate.

Each cluster in the FDCB routing algorithm has the potential to obtain gateway nodes, which maintain communications with adjacent clusters, via the following sequence of operations:

- Each idle node broadcasts a short beacon packet at periodic intervals containing its cluster ID announcing it is an active cluster member
- When a non-cluster member receives the packet it learns it can contact the neighboring cluster through that node
- The receiver sends a short beacon reply packet containing its cluster ID
- The two nodes are then gateway nodes which provide access to each others cluster

With the FDCB algorithm there is no need for the node aggregation process used in hierarchical routing algorithms since clusters are not required to be represented by any aggregated data structure. Consequently, the problem of erroneously aggregating a complex network into a single logical node is avoided.

Although the FDCB routing algorithm addresses many of the difficulties encountered with the traditional QoS routing methodologies (i.e., centralized and hierarchical) it employs a distributed routing method, which has significant negative performance implications as mentioned previously. More to the heart of this research, the paper does not talk to how failures that occur during a QoS connection are handled. Support for cluster joins and leaves is explicitly provided; however, the problem of mitigating impact on QoS in the event of an unpredicted node leave/failure is left untreated. It is therefore assumed that this event is handled by the common practice of rerouting the QoS traffic from the source.

Nargunam and Sebastian [22] illustrate the problems associated with conventional clustering techniques in which each cluster has exactly one node, the "cluster-head", which is responsible for organizing and establishing the cluster. This traditional cluster construction requires a cluster-head election scheme in which each time a cluster-head fails or leaves the cluster all available cluster nodes decide on a new cluster-head. This standard election scheme has a single point of failure, the cluster-head. That is, if the cluster-head fails or leaves the cluster, all information and responsibilities performed by the cluster-head become orphaned. To avoid this problem, Nargunam and Sebastian propose a fully distributed architecture in which clusters are created using a nontraditional cluster creation algorithm. In this new cluster creation algorithm, each cluster member maintains a QoS parameter table for each of its cluster members in addition to a table containing all gateway nodes in the cluster. In their efforts to avoid the problems of the traditional clustering scheme and improve scalability, FDCB is left with no effective way to handle connection failures. Also, the distributed routing design presented is ill-suited for the MANET QoS application. Although Nargunam and Sebastian provide a logical path in development of a solution to the problem of scalability with regard to QoS routing in an ad hoc mobile environment, it is believed that the FDCB routing protocol could be significantly improved.

2.7 Fault-tolerance in QoS Ad Hoc Networks

Chen and Nahrstedt [26] propose fault-tolerance techniques in an effort to reduce the impact on QoS disruptions due to link failures caused by network dynamics. It is important to note that Chen and Nahrstedt only consider applications which do not require hard guarantees. "Soft QoS means that there may exist transient time periods when the required QoS is not guaranteed due to path breaking or network partition" [26]. Further, Chen and Nahrstedt state that many multimedia applications accept soft QoS and use adaptation techniques to reduce the level of QoS disruption [6], [15], [28]. One technique presented is to repair the broken path at the node failed by shifting the traffic over to a neighboring node and then routing around the breaking point. This method avoids the costly process of rerouting the traffic from the source. The second technique involves using a multilevel *path redundancy* scheme. The idea is to establish multiple paths for the same connection. The *First-Level Redundancy* sends all data along all paths independently. This redundancy level is used for ‘critical’ QoS connections. The *Second-Level Redundancy* sends data along only the primary path and uses any secondary paths only in the event that the primary path is lost. This redundancy level is used for QoS connections which can tolerate a certain degree of QoS failure. The *Third-Level Redundancy* is similar to the second level except the secondary paths are not reserved; only calculated. If a failure should occur, an attempt will be made to reserve the secondary path.

The first technique, the repair algorithm, is proposed as the single approach to the following cases:

1) The source moves out of range of the first intermediate node in the path
2) An intermediate node moves out of range of either a preceding or successive node (preceding node is on the source’s side of the intermediate node, successive is on the destination’s side of the intermediate node)
3) The destination moves out of range of its preceding node (preceding node is the last intermediate node before the destination)
4) Any node in the path leaves the network

When case 2 occurs, the preceding node broadcasts a *repair-requesting* message to all its neighbors asking if any of them are able to take over the job of the defunct intermediate node. The neighbors that have links to the successive node reply their resource availabilities to the preceding node. If, based on the replies, the preceding node finds node *i* has sufficient resources for that role, it adds the link from itself to node *i* to the routing path and then sends *i* a *path-repairing* message. When *i* receives the *path-*

repairing message, it reserves the required resources and adds the link from itself to the successive node to the routing path. Once the path has been repaired, a *path-validation* message is generated to insure that the repaired path does not violate any of the end-to-end requirements. The *path-validation* message is sent to the destination which then sends the message to the source. The source then checks to see if the end-to-end requirements have been violated. If they have, the source will reroute the traffic or some QoS negotiation will take place with the user application. The performance metric used during simulation of the repair algorithm is the QoS ratio, defined as:

$$QoS\,ratio = \frac{total\,QoS\,time}{total\,QoS\,time + best - effort\,time}.$$

Where *best-effort time* is defined as the amount of time spent repairing the broken path. The x-axis is the mobility ratio, defined as:

$$mobility\,ratio = \frac{total\,moving\,time}{total\,stationary\,time + total\,moving\,time}.$$

The simulation results provided include a single graph which shows for a mobility ratio of less than 10%, the QoS ratio is above 95%. As expected, the QoS ratio decreases as the mobility ratio increases. For a mobility ratio more than 35%, the QoS ratio is below 80%. The conclusion is that Chen and Nahrstedt's routing algorithm should not be used in networks with high node mobility.

2.8 Summary

This chapter discussed the fundamental concepts and literature underlying the work presented in this thesis. The chapter began with a review of the QoS routing problem in wired networks and recent techniques that have been applied. The MANET was discussed as were implications of QoS in this mobile environment. A recent promising cluster based work aimed at providing QoS in the MANET was presented from the literature. Finally, two techniques for providing fault-tolerance in the QoS supporting MANET were considered. The first requires determining, and potentially implementing, multiple disjoint feasible paths [26]. This technique requires at least twice the number of network resources (depending on the redundancy level) in addition to increased computation time to determine multiple feasible paths for a single QoS connection. Hence, the efficiency of this method is exceedingly low. The second involves a packet broadcast process which can only be successful if the node responsible for performing the repair algorithm has a neighbor that can reach the successive node one hop away from the defunct (or failed) node[26]. This packet broadcast process, necessary for the repair algorithm proposed by Chen and Nahrstedt, involves communication overhead that is unnecessary if a cluster based fault-tolerant solution is used. Additionally, neither technique discusses the case where a failed link supported multiple QoS connections. Although nothing in the literature exists which directly addresses the problem of fault-tolerant QoS routing in hybrid mobile ad hoc networks, this chapter provides the foundation to develop the methodology discussed in Chapter Three.

III. Methodology

3.1 Chapter Overview

This thesis considers the fault-tolerant problem in H-MANETs designed to support QoS requirements. The previous chapter discussed two techniques that have been offered [26]; the first involving a path redundancy technique and the second concerning a local repair algorithm. The work presented here is similar to the local repair algorithm described by Chen and Nahrstedt [26] in which the underlying concept is to handle QoS connection failures at the site closest to the link break point. While the two protocols share this similarity, there are significant differences in the repair methods used and the level of fault-tolerance achieved. This chapter presents the motivation for, and explanation of, key design features of the extended fully distributed cluster based (EFDCB) routing protocol which is a fault-tolerant modification to the FDCB routing concept [22]. The definitions and assumptions used by the algorithm are also provided.

3.2 Problem Definition

3.2.1 Goals and Hypothesis.

Nargunam and Sebastian [22] propose a fully distributed algorithm, FDCB, in which a clustering technique is used to provide scalability by greatly lowering the amount of information which must be maintained at each node in the QoS supporting network. The FDCB algorithm addresses the MANET scalability problem successfully; however, it is missing the functionality necessary to maintain QoS connections when nodes supporting the QoS paths move, leave the network, or fail. Since Nargunam and

Sebastian provide no information on the matter, it is assumed that FDCB applies the commonly used rerouting procedure to the problem of broken QoS paths. That is, when a QoS path suffers a link breakage, the source is required to reroute the traffic via a completely new path. Additionally, FDCB in its current form uses a distributed reactive routing technique which causes undesired packet transmission latency, especially for the QoS routing application. Although FDCB does not provide a feasible routing scheme or local fault-tolerance it serves as groundwork to that end.

It is hypothesized that an extension to the FDCB protocol, namely EFDCB, will provide the scalability, efficiency, and fault-tolerance critical to maintain QoS connections in this mobile environment. The goal is to determine if the EFDCB QoS routing algorithm provides efficient QoS route recovery by testing it against the FDCB routing protocol. Note that the local fault-tolerant EFDCB algorithm only has to consider a fraction of the total number of network links when determining a new feasible path through the cluster. Hence, the burden of negotiating newly calculated QoS paths, as is done in the rerouting algorithm FDCB, is significantly reduced. For this reason, it is expected that this new local method will have a considerable runtime advantage resulting in improved QoS route recovery time. Faster QoS recovery time equates to lower QoS disruption time and therefore fewer dropped packets and improved throughput.

3.2.2 Approach.

To achieve efficient fault-tolerance, a cluster-head scheme is added to FDCB such that the cluster-head has complete 'cluster-state' knowledge. This means the cluster-head has connectivity awareness of all nodes in the cluster. Connectivity awareness includes knowledge of all QoS connections currently supported by each cluster member, each cluster member's resource availability and the cluster topology. With this scheme, when cluster node *i* leaves the cluster, due to mobility or failure, and the QoS paths supported by *i* are broken, the cluster-head has all information necessary to begin a re-negotiation process allowing the connection to be re-established with minimal delay if possible (i.e., if available cluster resources can support the QoS constraints of the failed connection). The cluster-head collects this knowledge by means of two processes: through communication with the other clusters in the system via a clustered FSR algorithm and by a local clustered information exchange algorithm. These processes ensure, with high probability and with low overhead, that knowledge of the systems' state is maintained both to repair existing paths and to initiate new connections.

3.3 System

3.3.1 Services.

The primary service provided by EFDCB is routing of QoS packets. The key to delivery of packets with required constraints is the underlying QoS routing algorithm; however, in the challenging MANET environment, links can break often and without warning. In this environment, the QoS routing algorithm needs a contingency plan for the eventual link breakage. This is where EFDCB exhibits its secondary service, and the one of most interest here – QoS disruption mitigation. When EFDCB is successful, packets are delivered such that the applications dependent upon the network are fully functional (e.g., the supported VoIP session has the desired end-to-end latency – hence, good voice quality is maintained throughout the conversation). Conversely, if the protocol fails the dependent applications could suffer lengthy QoS disruptions since the source will have to resort to rerouting.

3.3.2 Design.

3.3.2.1 Clustering.

Numerous schemes exist for clustering nodes in the MANET. FDCB constructs non-overlapping clusters based on bandwidth and delay factors of each link [22]. Another scheme uses location information obtained from the Global Positioning System to create a Virtual Grid Architecture (VGA) [1]. The purpose of the VGA is to cluster nodes into a fixed rectilinear virtual topology in an effort to make routing and network management as efficient as possible.

The fundamental clustering algorithm adopted here is a modified version of the Generalized Distributed and Mobility Adaptive Clustering (GDMAC) protocol from Ghosh and Basagni [14]. This clustering scheme was primarily chosen because it has been demonstrated to perform well when subjected to the three different mobility models (*random way point, random walk, Manhattan*) used in many prominent simulation studies of ad hoc networks [14]. Further, the protocol is straightforward, allowing the necessary modifications to be clearly illustrated and understood.

The modifications to GDMAC of interest to this research are those which make fault-tolerance in the QoS supporting MANET possible. It should be mentioned that the original Ghosh and Basagni clustering algorithm is not applied to QoS. This means an underlying QoS routing protocol as well as supporting procedures (i.e., for path negotiation, resource reservation and resource de-allocation) must be constructed. The QoS supporting procedures have been built into GDMAC via EFDCB.

The optimum cluster size is a parameter which is dependent on several characteristics of the MANET. Kumar and Gupta [17] show that the per node capacity of a random ad hoc network, where each node is capable of transmitting W bits per second is $\Theta\left(W / \sqrt{n \log n}\right)$ using a geometric analysis (where n is the number of nodes in the network). The cluster size used for this research leaves sufficient bandwidth for transmission of the required control packets (assuming each node has 54Mbps transmission rate for their 802.11 interface). Further, it is assumed clusters are situated such that the only inter-cluster nodes that are able to communicate are gateway nodes.

3.3.2.2 QoS Routing.

Since FDCB uses the on-demand reactive routing scheme, the decision was made to adopt a more proactive routing protocol for EFDCB. The QoS routing scheme used by EFDCB is the Clustered Fisheye State Routing (CFSR) protocol [12]. Unlike FSR, presented in Section 2.4.1, CFSR proposes a clustering framework in order to reduce redundant broadcast routing control messages. Recall that for FSR, the frequency at which node i sends its link state information to node j is dependent upon the distance from node i to node j (namely the *scope* node j falls in). The greater the distance, the less frequent the link state update.

CFSR allows clusterheads as well as gateway nodes to execute the original FSR algorithm sending out link state updates about the cluster while ordinary nodes are only allowed to send out link state information about themselves. This limits the number of messages sent from a significant portion of the network population (ordinary nodes). The result is lower overhead of the routing protocol. Assuming combinatorial stability, each node will become aware of the complete network state with a lower bandwidth cost. The disadvantage is that routing control messages traverse the network at a slightly lower rate since a much smaller fraction of network nodes are allowed to broadcast full control messages.

The authors of CFSR mention that with the scheme described above, redundancy is not minimized; however, it is reduced considerably. In order to minimize the redundancy it must be guaranteed that each clusterhead does not receive link state information about the same cluster from more than one gateway node. To accomplish this, the entire clustered network is partitioned into as many disjoint sets as the cluster has gateway nodes. These partitions are determined by finding the distance from each external network node to each local cluster gateway node using the topology graph stored in the routing table. The local cluster gateway node that has the shortest distance to the external network node includes that external node in its control message. Due to the previously described mechanics of FSR (Section 2.4.1), the gateway node closest to the external network node will be the first gateway node to receive the external nodes link state update. Hence, it will be responsible for providing this information to the cluster.

CFSR is QoS ready since all that needs to be changed in the current link state definition is the addition of bandwidth and channel quality information to the link entry. EFDCB uses CFSR as it is presented with few modifications. CFSR is initiated once the clustering algorithm converges. Before presenting the pseudo code for CFSR, some definitions are provided. Bolded text represents modifications to the original definitions.

CFSR Definitions

- *i* – the generic node executing the update procedure
- A_i – set of nodes that are adjacent to *i* (*i*'s neighbor list)
- *N* – set of external network nodes local gateway node *i* is closest to (*i* will send its updates to and receive updates from this set of nodes as described above)
- $TT_i.LS(j)$ – denotes the link state information reported by node *j*. **Link state info contains *j*'s *k* weights w_{jk} for the *k* QoS constraints(e.g., for three constrained routing, $k = 3$, w_{j1} could be the delay value, w_{j2} could be probability of packet loss, w_{j3} could be bandwidth) for each link *j* reports on**
- $TT_i.\ SEQ(j)$ – denotes the time stamp indicating the time node *j* has generated this link state information
- TT_i – *i*'s topology table. Each destination *j* has an entry in table TT_i for each QoS constraint **(e.g., if three constrained routing is used *j* would have $TT_i.LS(w_{j1})$, $TT_i.LS(w_{j2})$, $TT_i.LS(w_{j3})$).** Also, for each link state entry in TT_i *j* has $TT_i.SEQ(j)$.
- $NEXT_i$ – *i*'s next hop table. $NEXT_i(j)$ denotes the next hop to forward packets destined for *j* on the path with the required constraints
- Scope – defined as the set of nodes that can be reached within a given number of hops. (In the case discussed in Section 2.4.1, three scopes are used for 1, 2, and > 2 hops respectively.)
- D_i – $D_i(j)$ denotes the distance of the shortest path from *i* to *j*

Each node begins with an empty neighbor list A_i and an empty topology table TT_i. After node *i* initializes its local variables as shown in the *NodeInit*() procedure of Figure 3.1, it learns about its neighbors by examining the sender ID of each received packet. *i* then calls the *Pkt_process* procedure of Figure 3.2 on the received packet which contains the link state information received from its neighbors. The *Pkt_process* procedure ensures the most up to date link state information is used by comparing the local sequence number with the embedded sequence number *pkt.SEQ*(*j*). If any entry in the incoming message has a newer sequence number regarding destination *j*, $TT_i.LS(j)$ is replaced with *pkt.LS*(*j*) and $TT_i.SEQ(j)$ will be replaced by *pkt.SEQ*(*j*).

```
$proc\ Node(i) \equiv$
  $NodeInit(i);$
  $while\ TRUE\ do$
    $if\ PktQueue \neq \phi$   $//\ packet\ received$
      $foreach\ pkt \in PktQueue\ do$
        $A_i \leftarrow A_i \cup \{pkt.source\}$
        $Pkt\_process(i, pkt)$
      $od;$
    $fi$
    $CheckNeighbors(i);$
    $TT_i.LS(i) \leftarrow A_i;$
    $FindSP(i);$
    $RoutingUpdate(i);$
  $od$
.


   $proc\ NodeInit(i) \equiv$
     $foreach\ j \in V$
       $do$
       $A_i(j) \leftarrow \phi;$
       $D_i(j) \leftarrow \infty;$
       $NEXT_i(j) \leftarrow -1;$
       $SEQ_i(j) \leftarrow -1;$
     $od$
     $A_i \leftarrow A_i \cup \{x \mid link(i, x)\ exists\};$
     $TT_i.LS(i) \leftarrow A_i;$
     $D_i(i) \leftarrow 0;$
     $NEXT_i(i) \leftarrow i;$
     $t_i \leftarrow 0;$
     $SEQ_i(i) \leftarrow t_i;$
   .
```

Figure 3.1. Main (Node(*i*)) and Initialize (NodeInit(*i*)) Procedures of CFSR [12]

```
$proc\ Pkt\_process(i, pkt) \equiv$
  $source \leftarrow pkt.source;$
  $TT_i.LS(j) \leftarrow TT_i.LS(j) \cup \{source\};$
  $foreach\ j \in V$
    $do$
    $if\ (j \neq i) \wedge (pkt.SEQ(j) > TT_i.SEQ(j))$
      $then\ begin$
        $TT_i.SEQ(j) \leftarrow pkt.SEQ(j);$
        $TT_i.LS(j) \leftarrow pkt.LS(j);$
      $end$
    $fi$
  $od$
.


      $proc\ CheckNeighbors(i) \equiv$
        $foreach\ j \in A_i\ do$
          $if\ weight(i, j) = \infty$
            $A_i = A_i - \{j\};$
          $fi$
        $od$
      .
```

Figure 3.2. Packet Process and Check Neighbors Procedures of CFSR [12]

FindSP(*i*) (Figure 3.3) generates the shortest path tree rooted at *i*. The shortest-path algorithm used here is modified to generate a next hop table for each shortest path created. This shortest path tree is used by *i* to send route updates to the set of nodes in *N*.

The *RoutingUpdate*(*i*) procedure shown in Figure 3.4 scans through the topology table and if $D_i(x)$ is within range of the fisheye scope level *l*, $TT_i.LS(x)$ will be included in the update message. The $UpdateInterval_l$ attribute is used to adjust the link state update frequency for the various fisheye scopes.

```
$proc\ FindSP(i) \equiv$
    $// Dijkstra's\ shortest-path\ algorithm$
    $P \leftarrow \{i\};$
    $D_i(i) \leftarrow 0;$
    $foreach\ x \in \{ j \mid (j \in V) \wedge (j \neq i) \}\ do$
        $if\ x \in TT_i.LS(i)$
            $then\ D_i(x) \leftarrow weight(i,x);$
                $NEXT_i(k) \leftarrow k;$
            $else\ D_i(x) \leftarrow \infty; NEXT_i(k) \leftarrow -1;$
        $fi$
    $od$
    $while\ P \neq V\ do$
        $foreach\ k \in V-P, l \in P\ do$
            $Find\ (l,k)\ such\ that$
            $weight(l,k) = \min\{(D_i(l) + weight(l,k)\};$
        $od$
        $P \leftarrow P \cup \{k\};$
        $D_i(k) \leftarrow D_i(l) + weight(l,k);$
        $NEXT_i(k) \leftarrow NEXT_i(l);$
    $od$
.
```

Figure 3.3. Find shortest path procedure of CFSR[12]

```
$proc\ RoutingUpdate(i) \equiv$
  $t_i \leftarrow t_i + 1;$
  $TT_i.SEQ(i) \leftarrow t_i;$
  $TT_i.LS(i) \leftarrow \phi;$
  $foreach\ x \in A_i$
    $do$
    $TT_i.LS(i) \leftarrow TT_i.LS(i) \cup \{x\};$
  $od$
  $message.Senderid \leftarrow i;$
  $foreach\ x \in N\ do$
    $for\ ScopeLevel\ l := 1\ to\ L\ do$
      $if\ ((clock() \bmod UpdateInterval_l = 0)$
        $\wedge (D_i(x) \in FisheyeScope_l))$
        $then\ message.TT \leftarrow$
            $message.TT \cup \{TT_i.LS(x)\};$
      $fi$
    $od$
  $od$
  $broadcast(j, message)\ to\ all\ j \in A_i;$
.
```

Figure 3.4. Routing update procedure of CFSR[12]

Note that by adopting this new routing scheme over the original FDCB method of distributed routing, additional memory resources are required. Furthermore, although routing control packets are sent (as periodic updates are required), CFSR significantly reduces the amount of broadcast control packets over pure proactive protocols. This is because only a small fraction of the network population is allowed to broadcast full control messages. These changes allow each node to be aware of the complete network state with a lower bandwidth cost. Given the fact that the necessary link state information is now available, by implementing any efficient QoS routing algorithm (e.g., the limited path heuristic [30] discussed in Section 2.2.7), EFDCB is able to achieve its primary function of routing packets based on QoS constraints. Note that no restriction is imposed on the method of QoS routing employed by EFDCB.

3.3.2.3 Fault-tolerance.

Some definitions are now presented to enable a detailed description of the EFDCB fault-tolerant approach. An *intermediate* (*I*) *node* is any node that supports a QoS connection. A *defunct* (*D*) *node* is a cluster node that previously was an *I* node; but has either moved out range or has failed. A *gateway node* (*GWN*) is defined as a cluster node which is used to communicate with an adjacent cluster. A *potential GWN* (*P-GWN*) is a node which has the ability to communicate with the same adjacent cluster as the current *GWN*; however, it is only used in the event that the current *GWN* becomes a *D* node. As an example, in Figure 3.5 *n*3 is a *P-GWN* since it can communicate with *n*1 and the current *GWN* is *n*4 since it is currently communicating with *n*1. A *cluster-head* (*CH*) is a cluster node which has the responsibility of monitoring and updating a cluster table which records all QoS connections currently supported by the cluster. The *CH* is also responsible for initiating QoS connection repairs. Note that a *CH* can also be a *GWN*. An ordinary node is a node that is neither a *CH* nor a *GWN*.

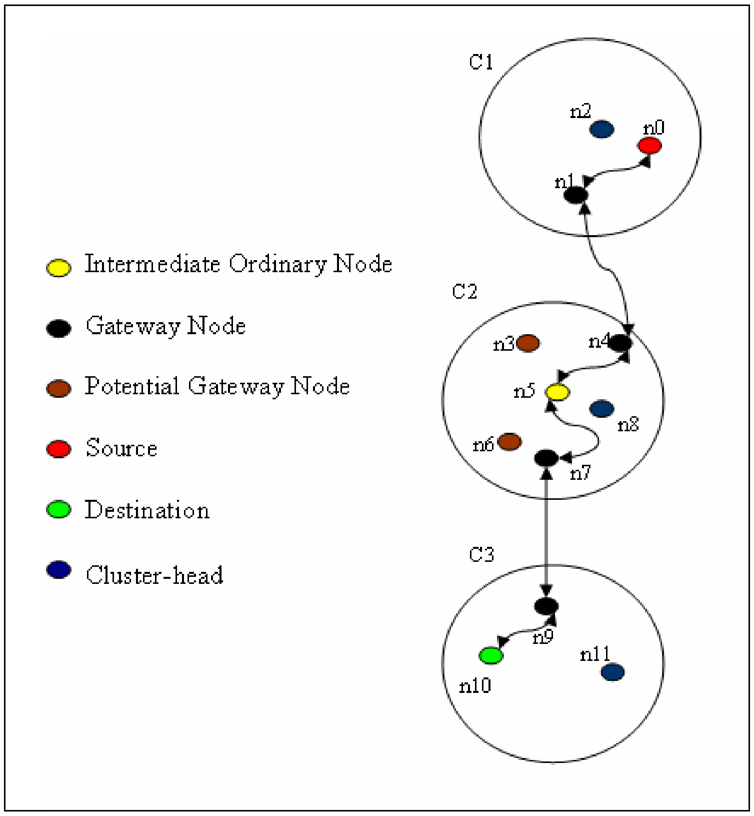


Figure 3.5. Clustered ad hoc network

Referring to Figure 3.5, let *n*0 be the source, *n*10 the destination and *P* the QoS path. Each node *i* in *P* has a successive (succ) node except *n*10. Further, each node *i* in *P* has a preceding (prec) node except *n*0. In a clustered ad hoc network such as the one described here, *P* can be broken if any of the following cases occur:

1) Ordinary *I* node moves out of range of a succ *I* node in the cluster(i.e., *n*5 moves out of range of *n*7)
2) Ordinary *I* node moves out of range of a prec *I* node in the cluster(i.e., *n*5 moves out of range of *n*4)

3) *I GWN* moves out of range of a succ *I* node in the cluster(i.e., *n*4 moves out of range of *n*5)
4) *I GWN* moves out of range of a prec *I* node in the cluster(i.e., *n*7 moves out of range of *n*5)
5) *I GWN* moves out of range of a succ *I GWN* in the cluster(i.e., if *n*4 were connected to *n*7 and *n*4 moves out of range of *n*7)
6) *I GWN* moves out of range of a prec *I GWN* in the cluster(i.e., if *n*4 were connected to *n*7 and *n*7 moves out of range of *n*4)
7) *I GWN* moves out of range of a succ *GWN* not in the cluster(i.e., *n*7 moves out of range of *n*9)
8) *I GWN* moves out of range of a prec *GWN* not in the cluster(i.e., *n*4 moves out of range of *n*1)
9) *I CH* moves out of range of a succ or prec *I* node in the cluster(i.e., if *P* were such that *n*8 had *n*4 as a prec node and *n*7 as a succ node and *n*8 moved out of range of either node)

In the case where the *D* node is an ordinary *I* node (cases 1 and 2), the cluster-head aggregates all available cluster resources as well as all cluster supported QoS routes and re-calculates the feasible QoS paths for the portion of the routes that traverse the cluster. Assuming the necessary resources for the QoS constraints of all paths exist; these new routes will be the optimum routes for the cluster given the current cluster topology. The necessary route resource negotiations are all handled within the cluster and the route is restored. In the case where the *D* node is a *GWN* (3-8), the *CH* first ensures the *P-GWN* can handle the QoS constraints previously supported by the now *D GWN*. Once it has been determined that the *P-GWN* can handle this traffic, the necessary resources are allocated. After a specified time of not receiving a beacon message from the *D GWN*, the preceding node to the *D GWN* attempts to route traffic through the *P-GWN* and the route is restored. Note: Case 9 (the *CH* fails while it is supporting a QoS connection) is addressed in Section 3.3.2.4.

With the EFDCB protocol, reducing the impact of a connection failure becomes more manageable since the cluster-head has complete cluster connectivity awareness. The result is a QoS route maintenance algorithm that accomplishes the goal of developing an efficient fault-tolerant QoS routing algorithm for the MANET.

3.3.2.4 EFDCB Protocol

Following the modifications and additions to the clustering algorithm and FDCB, EFDCB remains primarily message driven, as the separate algorithms were originally designed [14]. This indicates that the particular procedure executed by a node is dependent upon the message it received. Several types of messages are exchanged among the nodes. Before discussing the messages used in the EFDCB, the associated definitions and assumptions are presented. Note that in order to aid in distinguishing segments of the protocol, assumptions and definitions that are part of the original algorithms from those that are new, text will appear in bold whenever it represents a modification or addition to the original work.

EFDCB Definitions

- $\underline{v}$ – The generic node executing the algorithm (assume v encompasses the node's ID and its weight) [14]
- **Cluster-head – cluster node which has the responsibility of monitoring and updating the cluster QoS table as well as handling connection failures (i.e., aggregating cluster resources and supported QoS connection constraints, determining new feasible paths through the cluster, and notifying cluster nodes of the new paths)**
- **Gateway Node – cluster node that is used to communicate with an adjacent cluster**
- **Ordinary Node – cluster node which is neither a gateway node nor a Cluster-head**
- $\underline{w_v}$ – Weight of v, an integer > 0 which indicates how good that v is for serving as a cluster-head. For example, the weight could be computed based on the nodes residual bandwidth, available energy, or its mobility [14]
- ***GatewayNode*(–) – Boolean variable. *GatewayNode*(v) is set to true when cluster node v is adjacent to, and can communicate with, at least one other node in an adjacent cluster**
- H – The H parameter implements Ghosh and Basagni's idea that cluster re-organization is needed only when the new cluster-head is better than the current one by some specified value. That is, a clustered node switches to a newly arrived cluster-head only when the weight of the new cluster-head exceeds the

weight of its current cluster-head by a quantity *H*. By manipulating the value of *H*, the likelihood a node will switch to a new neighboring cluster-head can be controlled [14].

- <u>K</u> – Ghosh and Basagni's *K* parameter controls the spatial density of the cluster-heads. That is, up to $K \geq 0$ cluster-heads are allowed to be neighbors. $K = 0$ ensures that no two cluster-heads can be neighbors. By setting $K > 0$, the probability of cluster re-organization is lowered since a cluster-head is not forced to give up its position when up to *K*-1 cluster-heads with bigger weights become its neighbors[14]. Note: To simplify the EFDCB algorithm, $H = 0$ and $K = 0$ are the optimal values. With these values, the interaction of cluster-heads is avoided. That is, by setting $H = 0$, $K = 0$, cluster-heads are not allowed to be neighbors.
- <u>$\Gamma(v)$</u> – the set of all nodes one hop away from *v* in the same cluster [14]
- **<u>$\Pi(v)$</u> – the set of all nodes one hop away from *v* and in another cluster. Initialized to null and only updated if *v* becomes a Gateway node**
- ***<u>GT</u>* – table of gateway nodes for the cluster, maintained and broadcasted by the cluster-head**
- *<u>Cluster(v)</u>* – the set of nodes in *v*'s cluster, initialized to Ø [14]
- *<u>Cluster-head</u>* – the variable in which every node records the ID of the cluster-head that it joins (note *Cluster-head*$_v$ denotes the cluster-head of node *v*) [14]
- *<u>Ch(–)</u>* – Boolean variables. Node *v* sets *Ch*(*u*), $u \in \{v\} \cup \Gamma(v)$, to true when either it sends a CH(*v*) message ($v = u$) or it receives a CH(*u*) message from $u (u \neq v, u \in \Gamma(v))$ [14].

- ***NT*** **– node QoS table containing the current existing connections of the associated node (i.e., *v.NT* refers to node *v*'s own QoS table) – sent by node *v* to the cluster-head when either 1) node *v*'s cluster-head changes 2) node *v*'s QoS table changes or 3) update to the cluster-head is required**
- ***AT*** **– table containing the current available resources of the associated node (i.e., *v.AT* refers to node *v*'s own available resources table)**
- ***CT*** **– cluster QoS table containing the address, weight, *NT* and *AT* of all cluster nodes – knowledge shared by all cluster nodes via periodic cluster-head broadcast**
- *w*(*p*) – is the vector sum of all weights for all constraints of all edges in path *p*
- ***Connex*(–)** **– Boolean variable. Cluster-head sets *Connex*(*v*) to true when *v* is supporting a connection. *Connex*(*v*) is false otherwise**
- ***ConnexParams*(–)** **– table of variables into which the Cluster-head records the QoS connection requirements of a particular path (e.g., *ConnexParams*(*p*) would contain a table of the QoS parameters for path *p*)**
- ***PATH*(*dst*)** **– the set of QoS constrained paths to the destination dst (this set of paths could consist of only the single path which satisfies the collection of QoS constraints if a multiconstrained QoS routing algorithm is used or a set of split paths if a multicommodity flow implementation for QoS routing is used)**
- **$PATH_O$ – the original set of feasible paths through the cluster for the supported connections**

- ***$\underline{PATH_F}$*** **– the set of paths through the cluster that have failed due to the failed node**
- ***$\underline{PATH_N}$*** **– the newly calculated set of feasible paths through the cluster for the supported connections**

At cluster set up, or when a node is added to the network, its variables are initialized as follows:

Cluster-head = **NULL**

***Connex*(–)**, *Ch*(–), ***GatewayNode*(–)** = **false**

$PATH_O$, $PATH_F$, $PATH_N$, $\Gamma(v)$, $\Pi(v)$, *Cluster*(–) = **Ø**

***CT*, *GT*, *NT*, *AT*, *ConnexParams*(–)** = **NULL**

***H*, *K* = 0**

<ins>Assumptions</ins>

- All nodes have a unique identifier
- Two nodes can be cluster members of the same cluster if and only if their Euclidean distance is ≤ 30m (approximate range of 802.11g)
- Nodes signal their presence via a periodic beacon message and the drifting in of a new node is realized when its new neighbors hear its beacons
- When a node does not hear signals from a known neighbor within a certain amount of time, it assumes the neighbor to be either “dead” or out of range due to mobility

- Determining a node has failed or moved out of range will prompt the corresponding procedure
- All nodes have a single larger bandwidth interface (e.g., FSO transceiver, directional RF transceiver, etc.) for each node they can communicate with via 802.11 (the link state data is based on this link)
- All procedures are atomic **except the Route_traffic(*u*) procedure and the procedures executed for receipt of the PATH(*v.rsrcs*, *dst*) and CTS(*u*) messages – the following paragraphs will explain this in further detail**
- In order to discuss the relevant features of EFDCB, it is assumed that the CFSR algorithm has converged. That is, all gateway nodes in the network have path routing table entries for all network destinations. Also it is assumed that the associated applications using this QoS network have *soft* QoS constraints and use adaptive techniques to help minimize QoS disruptions as described by Chen and Nahrstedt [26] – discussed in Section 2.5. Combinatorial stability, as described in Chapter Two, is also adopted. Further, with this model, nodes have the ability to send and receive 802.11 best-effort traffic while sending and receiving QoS traffic along larger bandwidth, directional links. Finally, resources allocated for a QoS connection are de-allocated after a specified period of inactivity.

Messages

The message CH(v) is used by a node v to communicate to its neighbors that it intends to be a cluster-head [14]. The JOIN(v, u) message is sent by a node v to communicate to its neighbors that it will be part of the cluster whose cluster-head is node *u* [14]. The RESIGN(*w*) message is used to require the resignation from the role of cluster-head of any receiving cluster-heads whose weight is ≤ *w* [14]. **PATH(*v.rsrcs*, *dst*)** message is used by source *v* to request resources from each node *u* along the path of a new potential QoS connection to destination *dst*. The **CTS(*u*)** message is sent by destination *v* back to source *u* along the initialized (intermediately allocated) path to finalize the resource allocations. **PARAMS(*v.NT*, *v.AT*, *Cluster-head*)** message is used to send information about the supported connections (***v.NT)***, as well as the available resources (***v.AT)***, of node *v* to the cluster-head. **CLSTR_PRMS_UPDT(*v*, *CT*)** message is broadcast at regular intervals by cluster-head *v* to update each cluster nodes' cluster QoS table. **REPAIR(*ConnexParams*(*p*), *v*, *Cluster-head*)** message is sent by the cluster-head to notify node *v* to restore a connection using the information in the *ConnexParams*(*p*) table. **QOS_VALID(*u*, *v*)** message is sent from source *u* to destination *v* after a failed link has been repaired to ensure the end-to-end QoS constraints are sustained. The QoS validation message is initialized by source *u* after receiving **LINK_REPAIRED(*failed_node*, *v*, *u*)** from the node that is new to the path – *v*. **REPAIR_FAILURE(*v*, *Cluster-head*)** message is sent when *v*'s attempts to repair a failed connection also fails. **FAILED_CONNEX(*failed_node*, *p*, *v*)** message sent to source *v* of QoS path *p* in the event that the failed connection supporting *p* could not be repaired. **HELLO(*u*, *Cluster-head, Init*)** message used to create gateways via adjacent nodes in different clusters. At periodic intervals, node *u* sends a **HELLO(*u*, *Cluster-head, Init*)** message in an attempt to receive a **HELLO(*v*, *Cluster-head, Reply*)** message.

The procedures of the EFDCB protocol are presented in the paragraphs that follow. A few of the basic clustering procedures are largely the same as the original work; however, they are included for completeness. Most of the procedures are entirely new. As mentioned previously, in order to aid in differentiating segments and procedures of the protocol that are part of the original design from those that are new, pseudo code will appear in bold whenever it represents a modification or addition to the original work. The first procedure of interest is *Init*.

Init. The *Init* procedure remains predominantly as described by Ghosh and Basagni [14]. When the cluster is initialized, or when a node *v* is added to the network, *v* executes the *Init* procedure to determine its role. If among its neighbors there is a cluster-head with bigger weight, then *v* will join it and send a PARAMS message providing the cluster-head with *v*'s *NT* and *AT* which the cluster-head will then use to update the *CT*. If no node exists which has a weight bigger than *v*, *v* will be a cluster-head. In this case, the new cluster-head *v* checks the number of its neighbors that are already cluster-heads. If they exceed *K* = 0, then a RESIGN message is also transmitted, carrying the weight of the first cluster-head (the node with the lowest weight) that violates the *K*-neighborhood condition (this weight is determined by the operator $\min_K$). Since *K* = 0, the node with the largest weight will replace all cluster-heads within 30 meters who have a lower weight then *v*.

```
PROCEDURE Init;
begin
    if $\{z \in \Gamma(v) : w_z > w_v \wedge Ch(z)\} \neq \varnothing$  //if there exists a node z in the set of v's one hop
                                                  //neighbors st the weight of z > the weight of v
                                                  //AND either z has sent out a CH msg (v=z) or it
```

```
                                        //has received a CH msg from z (v≠z)
        then begin
                $x := \max_{w_z > w_v} \{z : Ch(z)\}$ ;//x=the node z with the max weight st v has
                                        //received a CH msg from z
                send JOIN(v,x):         //broadcast Join msg (I'm joining x's cluster)
                Cluster-head := x
                **send PARAMS(v.NT, v.AT, Cluster-head)**//send Cluster-head my QoS
                                                //table containing only my info
                end
        else begin                      //else, I'm going to be the new cluster-head
              send CH(v);               // send cluster-head msg
              Ch(v) := true;            // I sent the cluster-head msg = TRUE
              Cluster-head := v;        // cluster-head = ME
              Cluster(v) := {v} ;       // the set of nodes in my cluster is me
              if $\left|\{z \in \Gamma(v) : Ch(z)\}\right| > K$ then send RESIGN$\left(\min_K \{w_z : z \in \Gamma(v) \wedge Ch(z)\}\right)$
              // if the number of one hop nodes that are CH's is greater than K, send the
              // RESIGN msg //with the (K+1)th biggest weight
              end
end
```

Figure 3.6. Init Procedure

<u>Node_failure</u>. When node *v* is made aware of the failure of node *u*, *v* checks if its own role is cluster-head and if *u* was in its cluster. If this is the case, *v* removes *u* from *Cluster*(*v*). In this scenario, if *u* was an intermediate node supporting a connection (or multiple connections), cluster-head *v* aggregates all cluster resources and all supported QoS traffic and determines new feasible QoS paths. Node *v* then advises all relevant cluster nodes to support the required QoS connection via the REPAIR message. If all QoS connections cannot be supported by the cluster resources currently available (i.e., the cluster-head determined an infeasibility situation), the cluster-head sends a FAILED_CONNEX message to all sources that were using resources on the failed node and no path changes are implemented in the cluster. If *v* is an ordinary node, and *u* was

its own cluster-head supporting a QoS connection, and *v* is the node with the next weight w_v in descending order after w_u (i.e., $\exists v \forall z \in \{\Gamma(v) - \{u\}\} : w_v > w_z$), node *v* then becomes the new cluster-head and attempts to fix *u*'s failed connection. In the case where *v* is an ordinary node which is not the next weight w_v in descending order after w_u, and *u* was its own cluster-head (supporting or not supporting a QoS connection), *v* waits a specified period of time to receive the CH(*y*) message from the cluster node with the next highest weight to *u*, node *y*. If *v* receives the CH(*y*) message in the allotted time, *v* joins *y*'s cluster. If *v* does not receive the CH(*y*) message in the required time period, *v* must determine a new role for itself. In this case, *v* determines if there exists a cluster-head $z \in \Gamma(v) : w_z > w_v$. Node *v* then joins the cluster-head with the bigger weight and sends it's NT and AT to its new cluster-head, otherwise it becomes a cluster-head. In any case where *u* was the cluster-head and was also supporting a connection, the new cluster-head will attempt to repair *u*'s failed connection only if the cluster remains relatively preserved (i.e., the node next in weight after w_u becomes the new cluster-head). The re-clustering process is invoked for each node if node *v*, where $\exists v \forall z \in \{\Gamma(v) - \{u\}\} : w_v > w_z$, does not become the new cluster-head. The time required for re-clustering will add significant time to the connection recovery process; therefore, re-routing is employed in this situation. It is likely the cluster will change very little when a cluster-head fails since combinatorial stability is assumed. Further, since all cluster nodes are already aware of the next potential cluster-head (i.e., this information is broadcast by the cluster-head via the *CT* at periodic intervals), once a cluster-head fails cluster members wait a short amount of time (propagation delay + processing delay + error) to receive the CH(*v*) message from the expected new cluster-head. If the CH(v) message is not received in the allotted time, all cluster nodes must determine their new roles.

```
PROCEDURE Node_failure(u);
begin
    if Ch(v) and (u ∈ Cluster(v))
        then    Cluster(v) := Cluster(v) − {u}
            if Connex(u) = true then
                if $\left\{ \sum_{\forall z \in Cluster(v)} (z.NT + z.AT) \geq \sum_{\forall p \in PATH_O} (ConnexParams(p)) \right\}$
                    then begin
                        calculate $PATH_N$
                            for each path p in $PATH_N$
                                for each node n in p
                                    send REPAIR(ConnexParams(p), n, Cluster-head)
                fi //connection cannot be repaired due to resource limitations
                else for each p in $PATH_F$
                    for each source node s of p
                        send FAILED_CONNEX(failed_node, p, s)
            fi //no broken connection to repair
    else if Cluster-head = u then
        if $\exists v \forall z \in \{\Gamma(v) - \{u\}\} : w_v > w_z$
            begin   //I'm going to be the new cluster-head
            send CH(v);// send cluster-head msg
            Ch(v) := true;       //v sent the cluster-head msg = TRUE
            Cluster-head := v;  // cluster-head = ME
            Cluster(v) := Cluster(u) − {u}; //current set of nodes in my cluster
            if $|\{z \in \Gamma(v) : Ch(z)\}| > K$ then
                send RESIGN$(\min_K \{w_z : z \in \Gamma(v) \wedge Ch(z)\})$
            fi
            if Connex(u) = true then
                if $\left\{ \sum_{\forall z \in Cluster(v)} (z.NT + z.AT) \geq \sum_{\forall p \in PATH_O} (ConnexParams(p)) \right\}$
                    then begin
                        calculate $PATH_N$
                            for each path p in $PATH_N$
                                for each node n in p
                                    send REPAIR(ConnexParams(p), n, Cluster-head)
                fi //connection cannot be repaired due to resource limitations
                else for each path $p$ in $PATH_F$
                    for each source node $s$ of $p$
                        send FAILED_CONNEX($failed\_node$, $p$, $s$)
            fi //no broken connection to repair
            end
        else wait($t$ time units)
            if received CH($y$) == true
                then begin
                    send JOIN($v$,$y$):
                    Cluster-head := $y$
                    send PARAMS($v.NT$, $v.AT$, Cluster-head)
                    end
            else if $\{z \in \Gamma(v) : w_z > w_v \wedge Ch(z)\} \neq \emptyset$
                then begin
                $x := \max_{w_z > w_v} \{z : Ch(z)\}$
                send JOIN($v$,$x$):
                Cluster-head := $x$
                send PARAMS($v.NT$, $v.AT$, Cluster-head)
                end
            else begin   //else, I'm going to be the new cluster-head
                send CH($v$);// send cluster-head msg
                $Ch(v)$ := true;    //sent the cluster-head msg = TRUE
                Cluster-head := $v$;  // cluster-head = ME
                Cluster($v$) := $\{v\}$;  //current set of nodes in my cluster is me
                if $|\{z \in \Gamma(v) : Ch(z)\}| > K$ then
                    send RESIGN$(\min_K \{w_z : z \in \Gamma(v) \wedge Ch(z)\})$
                fi
                end
end
```

Figure 3.7. Node_failure Procedure

New_link. The *New_link* procedure is largely the same as Ghosh and Basagni's original design except for the addition of the send PARAMS message. Once cluster node *v* discovers a new node *u*, it first checks to see if *u* is a cluster-head. If *u* is a cluster-head and weight w_u is greater than the weight of *v*'s current cluster-head, *u* becomes *v*'s new cluster-head and sends *u* a PARAMS message. Conversely, if *v* is a cluster-head and the number of its neighboring cluster-heads is greater than 0, the weight of the cluster-head *x* that violates the $K = 0$ condition is determined. If it turns out that $w_v > w_x$, node *x* will be sent the RESIGN message. If there is no cluster-head *x* such that $w_v > w_x$, *v* will no longer be a cluster-head and will join the cluster-head with the biggest weight. Node *v* will then send its newly acquired cluster-head the PARAMS message.

```
PROCEDURE New_link(u);
begin
    if Ch(u) then
        if ($w_u > w_{Cluster\text{-}head} + H$)
            then begin
                send JOIN(v, u);
                Cluster-head := u;
                send PARAMS(v.NT, v.AT, Cluster-head);
                if Ch(v) then Ch(v) := false
                fi
            end
        else if Ch(v) and $\left|\{z \in \Gamma(v) : Ch(z)\}\right| > K$ then
            begin
                $w := \min_K \{w_z : z \in \Gamma(v) \wedge Ch(z)\}$;
                if $w_v > w$ then send RESIGN(w)
                else begin
                    $x := \max_{w_z > w_v} \{z : Ch(z)\}$;
                    send JOIN(v, x);
                    Cluster-head := x;
                    send PARAMS(v.NT, v.AT, Cluster-head);
                    Ch(v) := false
                end
            end
end
```

Figure 3.8. New_Link Procedure

Route_traffic(u). Source node *v* is made aware of the need to route new traffic by the associated application. Node *v* checks its cluster members to see if *u* is in this set of nodes. If *u* is in the current cluster, *u*'s available resources are obtained from the *CT* table. If the required resources are available, they are reserved and the traffic is sent. If the destination is not in the current cluster, *v* forwards the PATH(*v.rsrcs*, *dst*) to the cluster gateway nodes.

```
PROCEDURE Route_traffic(u);
begin
    if $\left(\left\{z \in \Gamma(v) : z = u\right\}\right)$ then
        if (v.rsrcs ≤ u.AT ) then
            $x := address_u\left\{z \in CT : z = u\right\}$
            $y := address_v\left\{z \in CT : z = v\right\}$
            send PATH(y.rsrcs, dst)
            counter = t time units
            while (CTS(u) not received OR counter ≠ 0) do
                counter--
            od
            if (counter == 0 AND CTS(u) not received) then
                exit(1) //destination is not responding
            else if (CTS(u) received)
                send DATA(x, y)
    else {
        for each gateway node n
            $x := address_u\left\{z \in CT : z = n\right\}$
            $y := address_v\left\{z \in CT : z = v\right\}$
            send PATH(y.rsrcs, dst)
        next n
end
```

Figure 3.9. Route_traffic Procedure

Note: the following procedures are initiated when the corresponding message is received.

On receiving PARAMS(*u.NT*, *u.AT*, *Cluster-head*): performed by the *Cluster-head* (in this case the cluster-head is node *v*) – On receiving the message PARAMS(*u.NT*, *u.AT*, *v*), Cluster-head *v* updates the *CT* with this new information. *v* then checks the time since the broadcast of the last *CT*. If sufficient time has passed since the last broadcast, *v* broadcasts the *CT* to all cluster nodes. During construction of a new cluster, the cluster-head has the potential to receive many *CT* updates. The *CT* is never sent at intervals less than some time *T*. The effect is to allow the cluster to reach a level of stability before broadcasting cluster table updates. Although it is not explicitly shown in pseudo code, the CLSTR_PRMS_UPDT(*CT*, *v*) message is broadcast at regular intervals (similar to the beacon message) throughout the life of the cluster-head.

```
On receiving PARAMS(u.NT, u.AT, Cluster-head);
begin
    if Ch(v) AND v = Cluster-head then begin
                $CT = CT \cup \{u.NT\} \cup \{u.AT\}$
                T = current_time – previous_CT_send_time
                if (T > specified_ interval_time){
                    broadcast CLSTR_PRMS_UPDT(CT, v);
                    previous_CT_send_time = current_time;
                }
                end
end
```

Figure 3.10. PARAMS Message Receipt (for the cluster-head)

<u>On receiving PARAMS(*CT*, *Cluster-head*)</u>: On receiving the message PARAMS(*CT*, *Cluster-head*), *v* first ensures that the cluster-head which sent the message is *v*'s cluster-head. Node *v* then checks to see that the *CT* has an accurate account of *v*'s *NT* and *AT*. If these two conditions hold, *v* records the received cluster QoS table. Otherwise, if *v* received from the correct cluster-head but *CT* is incorrect, *v* sends its *NT* and *AT* to the cluster-head.

```
On receiving PARAMS(CT, Cluster-head);
begin
  if $Cluster\text{-}head_v$ = Cluster-head then begin
```

```
        if($v.NT == NT_v \in CT$ AND $v.AT == v.AT \in CT$)
            $CT_v = CT$
        else{
            send PARAMS($v.NT$, $v.AT$, Cluster-head)
            }
                            end
end
```

Figure 3.11. PARAMS Message Receipt (for non-cluster-head nodes)

<u>On receiving REPAIR(*ConnexParams*(*p*), *v*, *Cluster-head*):</u> On receiving the message REPAIR(*ConnexParams*(*p*), *v*, *Cluster-head*), *v* first ensures that the cluster-head which sent the message is *v*'s cluster-head. Node *v* then checks to see that it has the resources to support the new connection described in the *ConnexParams*(*p*) message. Upon verification of available resources, *v* uses the information it knows about the connection (contained in the *ConnexParams*(*p*) table) to attempt to restore the link. If the link is restored successfully, LINK_REPAIRED(*failed_node*, *v*, *u*) is sent from *v* to source *u* of the QoS traffic. If *v* determines that it cannot communicate with the nodes necessary to make the connection, *v* sets the boolean ERROR to true, and *v.NT* and *v.AT* are sent to the cluster-head whether the connection is reconnected or not. Notice that this procedure is executed for reconnection of links that may or may not have failed since the cluster-head determines a new set of feasible paths for all connections that traverse the cluster when a failure occurs (assuming the necessary cluster resources are available).

```
On receiving REPAIR(ConnexParams(p), v, Cluster-head);
begin
        if $Cluster\text{-}head_v == Cluster\text{-}head$ then
            if ($v.AT < ConnexParams(p)$) then
                send PARAMS($v.NT$, $v.AT$, Cluster-head);
                send REPAIR_FAILURE($v$, Cluster-head)
        else {                    //install new connection
```

```
            v.AT = v.NT - ConnexParams(p)
            if(!ERROR){        //if there was no problem making the nec connections
                send PARAMS(v.NT, v.AT, Cluster-head);
                Connex(v) = true
                LINK_REPAIRED(failed_node, v, u)
            else
                send REPAIR_FAILURE(v, Cluster-head)
end
```

Figure 3.12. REPAIR Message Receipt

On receiving **LINK_REPAIRED(*failed_node*, *u*, *x*)**: **This concept is borrowed from** Chen and Nahrstedt [26]. On receiving the message LINK_REPAIRED(*failed_node*, *u*, *x*), source node *v* sends the QOS_VALID(*y*, *v*) to any destination node *y* which received QoS traffic that passed through *failed_node*. Once *y* receives this message, it sends the QOS_VALID(*v*, *y*) message enabling *v* to determine if end-to-end delay constraint has been violated.

```
On receiving LINK_REPAIRED(failed_node, u, x);
begin
        for each route r through failed_node
            for each destination t of r
                send QOS_VALID(t, v)
            next t
        next r
end
```

Figure 3.13. LINK_REPAIRED Message Receipt

On receiving **REPAIR_FAILURE(*u*, *Cluster-head*):** performed by the *Cluster-head* only: On receiving the message REPAIR_FAILURE(*u*, *Cluster-head*) cluster-head *v* immediately sends a FAILED_CONNEX(*failed_node*, *v*, *x*) back to any source *x* which was using resources on *failed_node*.

```
On receiving REPAIR_FAILURE(u, Cluster-head);
```

```
begin
    for each source s using failed_node
        send FAILED_CONNEX(failed_node, v, s);
    next s
end
```

Figure 3.14. REPAIR_FAILURE Message Receipt

On receiving **FAILED_CONNEX(*failed_node*, *u*, *v*):** On receiving the message FAILED_CONNEX(*failed_node*, *u*, *v*), node *v* attempts to reroute the associated QoS traffic via the Route_traffic(*t*) procedure for each route *r* which traversed *failed_node*.

```
On receiving FAILED_CONNEX(failed_node, u, v);
begin
    for each route r that traversed failed_node
        for each destination t of r
            Route_traffic(t)
        next t
    next r
end
```

Figure 3.15. FAILED_CONNEX Message Receipt

On receiving **QOS_VALID(*v*, *u*)**: On receiving the message **QOS_VALID(*v*, *u*),** node *v* immediately sends a **QOS_VALID(*u*, *v*)** back to the source.

```
On receiving QOS_VALID(v, u);
begin
    send QOS_VALID(u, v)
end
```

Figure 3.16. QOS_VALID Message Receipt

On receiving HELLO(*u*, *Cluster-head, Init*): On receiving the message HELLO(*u*, *Cluster-head, Init*), node *v* checks the value of *Cluster-head* to determine whether the sender, *u*, is a cluster member of the same cluster or the member of an adjacent cluster. If *u* is a cluster member of the same cluster the message is discarded. If the sender is the member of an adjacent cluster *v* then checks to see if it has received a HELLO message from *u* previously. If *v* has not received a HELLO message previously from *u*, *v* notes that it can contact the adjacent cluster through node *u*. Node *v* then notifies the cluster-head of this, and transmits a HELLO(*v*, *Cluster-head*, *Reply*) message to the sender. *u* receives the HELLO(*v*, *Cluster-head*, *Reply*) message, notes that it can contact the adjacent cluster through *v*, and notifies its cluster-head. Node *v* is now a gateway node from its cluster to *u*'s cluster and *u* is now gateway node from its cluster to *v*'s cluster.

```
On receiving HELLO(u, Cluster-head, Init or Reply);
begin
    if Ch(v) and $Cluster\text{-}head_v \neq Cluster\text{-}head$ then begin
            if Init then
                if $u \notin \Pi(v)$ then
                    $\Pi(v) = \{u\} \cup \Pi(v)$
                    send MYNGHBRS($\Pi(v)$, Cluster-head)
                    send HELLO(v, Cluster-head, Reply)
                    GatewayNode(v) = true
                fi
            else if Reply then
                if $u \notin \Pi(v)$ then
                    $\Pi(v) = \{u\} \cup \Pi(v)$
                    send MYNGHBRS($\Pi(v)$, Cluster-head)
                    GatewayNode(v) = true

                fi
            fi
        end
end
```

Figure 3.17. HELLO Message Receipt

On receiving MYNGHBRS($\Pi(u)$ *Cluster-head*): performed by the *Cluster-head* only: On receiving the message MYNGHBRS($\Pi(u)$, *Cluster-head*), *Cluster-head* amalgamates the received $\Pi(u)$ into the cluster gateway table (*GT*). *Cluster-head* then broadcasts the *GT* to all cluster nodes. Since it is unlikely that MYNGHBRS messages will be transmitted often, no limitation is imposed on the frequency with which the CLSTR_GN_UPDT(*GT*, *v*) messages can be sent.

```
On receiving MYNGHBRS($\Pi(u)$, Cluster-head);
begin
    if $Ch(v)$ and $v$ = Cluster-head then begin
                            $GT = GT \cup \Pi(u)$
                            broadcast CLSTR_GN_UPDT($GT$, $v$);
                            end
end
```

Figure 3.18. MYNGHBRS Message Receipt

On receiving PATH(*u.rsrcs*, *dst*): On receiving the message PATH(*u.rsrcs*, *dst*), *v* checks to see if it has the required resources using its availability table (*v.AT*). If not, *v* drops the PATH packet. If *v* has the required resources, it does an intermediary allocation of the requested resources (adjusting the *AT* to reflect this potential additional connection). If *v* is the destination, it allocates the necessary resources and responds with a CTS(*u*) message which traverses back down the path to source *u*. Node *v* then waits a predetermined amount of time to receive the data packet. If the data packet is not received in this allotted time, *v* de-allocates the requested resources. In the case where *v* is not the destination, once the intermediary resource allocation is done, a count down timer is initiated. If the associated CTS(*u*) message is not received before the counter expires, the resources are de-allocated. Note that this procedure is not atomic since intermediary nodes must be looking for the receipt of the CTS(*u*) message or handling other received messages or events while decrementing its counter. By using the count down timer, resources are not held for extended periods in the event that the path is never used (e.g., if a node farther down the path cannot support the QoS request). If *v* is not the destination, *v* checks to see if the destination is in its routing table and if the path in routing table entry meets the required constraints. If so, *v* routes the PATH(*u.rsrcs*, *dst*) message on to the destination. If no path exists to the destination which can support the required constraints, *v* discards the packet.

```
On receiving PATH(u.rsrcs, dst);
begin
        if ($u.rsrcs \le v.AT$ ) then
            $v.AT = v.AT - u.rsrcs$
            if ($v == dst$) then
                send CTS(u)
                counter = t time units
                while(DATA(u, v) not received AND $counter \neq 0$) do
                    counter––
                od
                if (counter == 0 AND DATA(u, v) not received) then
                    $v.AT = v.AT + rsrcs$          //de-allocate after waiting t time units
                fi

            else
                flag = 0;
                $x := address_{destination} \{z \in CT : z = dst\}$
                for path p in PATH(x)
                    if ($w(p) < u.rsrcs$)
                        send PATH(u.rsrcs, dst)
                        flag = 1;
                        break
                if(flag == 1) then
                    counter = t time units
                    while (CTS(u) not received $\wedge$ $counter \neq 0$) do
                        counter--
                    od
                    if (counter == 0 AND CTS(u) not received) then
                        v.AT = v.AT + rsrcs//de-allocate after waiting t time units
                    fi
                fi
            else
                //drop the packet
end
```

Figure 3.19. PATH Message Receipt

On receiving the CH(u): The procedure executed upon receipt of the CH(u) message is essentially the same as the original associated procedure. The only exception is the addition of the two send PARAMS lines which are executed after a node accepts a new cluster-head. When v's neighbor u becomes a cluster-head and v receives the CH message from node u, v checks to see if w_u is larger than the weight of v's current cluster-head (plus the parameter H, which equals 0 throughout this thesis as mentioned in the definitions portion of this section). If it is, v joins u's cluster. If, on the other hand, v is a cluster-head such that it has more than K neighbors which are clusters (K also equals 0 as mentioned in the definitions portion of this section), the cluster-head with the smallest weight is determined so that it may give up its cluster-head position.

```
On receiving CH(u);
begin
        if ($w_u > w_{Cluster\text{-}head} + H$) then
            send JOIN(v, u);
            Cluster-head := u;
            send PARAMS(v.NT, v.AT, Cluster-head);
            if Ch(v) then
                Ch(v) := false
                return
            else if Ch(v) and $\left|\{z \in \Gamma(v) : Ch(z)\}\right| > K$ then
                $w := \min_K \{w_z : z \in \Gamma(v) \wedge Ch(z)\}$;
                if $w_v > w$ then
                    send RESIGN(w)
                else
                    $x := \max_{w_z > w_v} \{z : Ch(z)\}$;
                    send JOIN(v, x);
                    Cluster-head := x;
                    send PARAMS(v.NT, v.AT, Cluster-head);
                    Ch(v) := false
end
```

Figure 3.20. CH Message Receipt

On receiving CTS(*u*): On receiving CTS(*u*) message, *v* checks to see if it is source *u*. If it is, it begins transmission of the QoS traffic. If not, and *v* has earmarked resources for *u*'s connection, the intermediate resource allocation that was done previously is finalized and the node table (*v.NT*) is updated to reflect the newly supported connection. *v* then transmits this update to the cluster-head.

```
On receiving CTS(u);
begin
    if myID = u then
        send QoS traffic
    else if (PATH(u.rsrcs, dst) received $\wedge$ counter $\neq$ 0) then
        v.NT = v.NT $\cup$ u.rsrcs
        send PARAMS(v.NT, v.AT, Cluster-head)
```

Figure 3.21. CTS Message Receipt

On receiving JOIN(*u*, *z*): The procedure executed upon receipt of the JOIN(*u*, *z*) message is essentially the same as the original JOIN(*u*, *v*) handling procedure. The only exception to this is the addition of the send PARAMS line executed after a node accepts a new cluster-head. After having received the JOIN(*u*, *z*) message, the behavior of node *v* depends on whether it is a cluster-head or not. If *v* is a cluster-head, it has to check for one of two cases. Node *v* checks if *u* is joining its cluster (i.e., $z = v$) or if *u* belonged to its cluster and is now joining another cluster (i.e., $z \neq v$). In the first case, *u* is added to *Cluster*(*v*) and then sends its *NT* and *AT* to the cluster-head. In the second case, *u* is removed from *Cluster*(*v*). If *v* is not a cluster-head and *u* was its cluster-head *v* has to determine its role. That is, it will join a new cluster-head *x* such that $w_x > w_v$ if such an *x* exists. Else it will become a cluster-head and ensure the *K*-neighborhood parameter is respected.

```
On receiving JOIN(u, z);
begin
    if Ch(v) //if I am the cluster-head (i.e., if v is the cluster-head)
        then if z = v then Cluster(v) := Cluster(v) ∪{u} // I (v) am u's new CH
            else if u ∈ Cluster(v) then Cluster(v) := Cluster(v) – {u}
    else if Cluster-head = u then //if u was my CH and it is now making z its CH
            if $\{z \in \Gamma(v) : w_z > w_v \wedge Ch(z)\} \neq \varnothing$
                then begin
                    $x := \max_{w_z > w_v} \{z : Ch(z)\}$
                    send JOIN(v, x):
                    Cluster-head := x
                    send PARAMS(v.NT, v.AT, Cluster-head)
                    end
            else begin   //else, I'm going to be the new cluster-head
                    send CH(v);// send cluster-head msg
                    Ch(v) := true;   //sent the cluster-head msg = TRUE
                    Cluster-head := v;  // cluster-head = ME
                    Cluster(v) := {v} ; //set of nodes in my cluster is me
                    if $|\{z \in \Gamma(v) : Ch(z)\}| > K$ then
                        send RESIGN$(\min_K \{w_z : z \in \Gamma(v) \wedge Ch(z)\})$
                end
end
```

Figure 3.22. JOIN Message Receipt

On receiving RESIGN(*w*): The procedure executed upon receipt of the RESIGN(*w*) message is essentially the same as the original procedure. The only exception to this is the addition of the send PARAMS line which is executed after a node joins a new cluster. After having received the RESIGN(*w*) message, node *v* checks if $w_v \leq w$. If this condition is met, *v* gives up its cluster-head position and joins the nearest cluster-head with the larges weight. Once *v* has received the RESIGN message and confirmed the need for its resignation, it sends its *NT* (supported connections list, as well as the available resources) to the new cluster-head.

```
On receiving RESIGN($w$);
begin
    if $Ch(v)$ and $w_v \leq w$ then begin
        $x := \max_{w_z > w_v} \{z : Ch(z)\}$;
        send JOIN($v$,$x$);
        $Cluster\text{-}head := x$;
        send PARAMS($v.NT$, $v.AT$, $Cluster\text{-}head$)
        $Ch(v)$ := false
        end
end
```

Figure 3.23. RESIGN Message Receipt

3.4 Summary

This chapter presents key design features of the EFDCB routing protocol. The EFDCB protocol is the unification of the modified GDMAC and FDCB algorithms which uses CFSR for QoS routing. The approach of EFDCB is to implement a cluster-head model to mitigate connection failures. This cluster-head model employs a cluster state knowledge sharing process which has the fundamental objective of making the cluster-head aware of current supported QoS connections in the cluster. The cluster state knowledge is also shared with all cluster members to assist in the event of a cluster-head failure. Additionally, CFSR allows each node in the EFDCB network to be aware of the complete network state with low bandwidth impact. EFDCB is different than FDCB since link failures can now be handled locally instead of rerouting the traffic from the source and QoS traffic can now be routed with lower packet transmission delay. The EFDCB fault-tolerant method differs from Chen and Nahrstedt's [26] work in that here a clustered approach is used which helps to avoid the limitation of Chen and Nahrstedt's repair algorithm. Specifically, EFDCB doesn't require that the predecessor of the failed node be capable of reaching the failed node's successor – this limitation of Chen and Nahrstedt's repair algorithm was discussed in Section 2.7. Consequently, EFDCB protocol presents an innovative solution to fault-tolerance in the QoS supporting MANET.

In summation, the local technique used by EFDCB is expected to be more efficient than FDCB's global paradigm at handling failures. In order to contrast these competing methods, Chapter Four presents the concerned performance metrics, evaluation technique, experimental design and data analysis. Finally, Chapter Four provides an interpretation of the results.

IV. Analysis and Results

4.1 Chapter Overview

This chapter documents a study of how the rate of node failures affect recovery time, number of dropped packets, throughput and amount of sustainable flow bandwidth for the EFDCB and FDCB protocols. The experimental design developed for this study describes 22 experiments. These experiments are aimed at generating data which provides insight to the competing protocols' ability to handle link failures.

The goal of the data analysis is to establish the performance of the EFDCB protocol compared to its predecessor FDCB when subjected to a network experiencing random node failures of varying failure rates. Since the hypothesis is that EFDCB will provide efficient fault-tolerance over FDCB in the failure prone mobile environment, the analysis shows how the key metrics are affected by network connection failures. Analysis of the data collected from the experiments is presented in this chapter. Finally, the data is interpreted and conclusions are drawn.

4.2 System Boundaries

This research does not focus on the ability of the EFDCB protocol to lower resource requirements [22]; rather, it focuses on the EFDCB protocol's ability to lower broken connection recovery time for QoS traffic. The most vital component of the system is the QoS routing protocol. Nevertheless, since connection recovery time improvement is the focus of this research, the component under test (CUT) is the fault-tolerant portion of the EFDCB system.

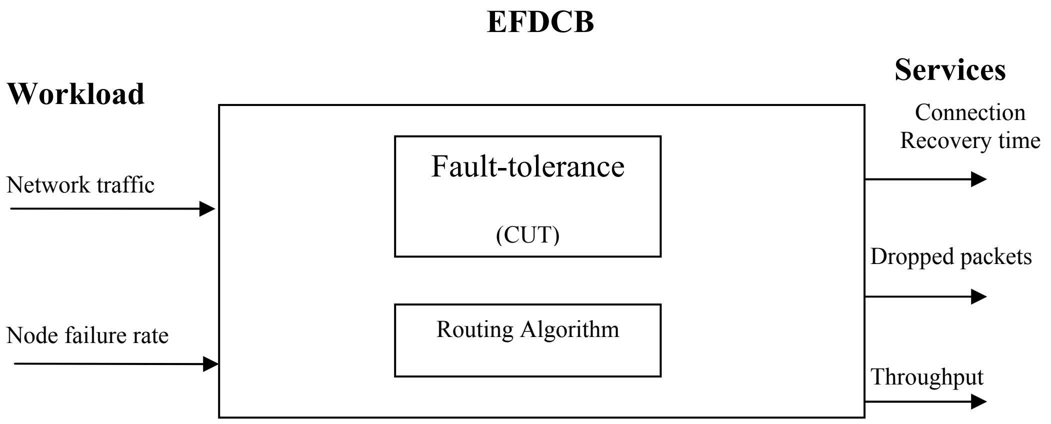


Figure 4.1. An Illustration of the EFDCB Protocol

Given the focus of this research, the intent is not to implement all of the developed EFDCB protocol, but the salient features required to determine if the EFDCB QoS routing algorithm provides efficient QoS route recovery. For this reason, the clustering portion of EFDCB algorithm is 'boot-strapped'. That is, it is assumed that the MANET is already clustered when the system initializes. Further, since clustering is hardcoded, node *i* moving out of communication range of node *j* is simulated by forcing *i* to fail. CFSR is also boot-strapped in this simulated system by using a (centralized) QoS routing algorithm which employs source routing based on a single constraint, namely bandwidth. The algorithm models traffic requests as multicommodity flows, discussed in Section 2.2.5, to determine if the traffic bandwidth demands can be satisfied. Note that with this routing model all nodes have complete network state knowledge as with CFSR; however, since network traffic is modeled as multicommodity flows, traffic flows (a flow of data from a source to a destination) can be split. Simulations were run on ns2 using a combination of custom middleware application agents and a custom routing module to emulate EFDCB. For simplicity, a wired scenario was used in ns2. The H-MANET is simulated by using a method in which lower bandwidth is used for routing control and beacon packets (i.e., 54Mbps omni-directional links) and higher bandwidth (i.e., 200-100Mbps directional links) for data traffic. The idea is to simulate each node having a higher bandwidth interface for every node it can communicate with via lower bandwidth wireless technology. Mobility was emulated by causing links to fail or recover at appropriate times. Similarly, failures were simulated by causing nodes in the system to go off-line at appropriate times.

4.3Workload

The system workload is the rate at which nodes in the network fail. Hence, a different node failure rate value is used for each group of simulations of the experimentation. The particular node that fails is chosen randomly. The number and type of QoS connection requests made by each source node is kept constant (as are the number of source-destination pairs and length of connections) during the simulations; however, these values are changed between experimental phases. The intra-cluster bandwidth is also altered between experimental phases. This is explained in more detail in the experimental design discussion. Since the EFDCB algorithm is tested against the original FDCB protocol, the same workloads are used with both algorithms.

4.4 Performance Metrics

The metrics for measuring system performance in this analysis are connection recovery time, number of dropped packets, throughput, and amount of sustainable flow bandwidth. With these data the comparative strengths and weaknesses of these competing algorithms can be determined. Connection recovery time is defined as the interval of time required to reestablish a failed connection from the moment data traffic stops. More specifically, the recovery time begins the moment the connection fails and ends when the destination receives the next data packet. This time interval measures how efficient the routing algorithms are at repairing broken links. With this knowledge, conclusions can be made about how well the offered traffic load is serviced. Traffic is expected to be serviced at a higher rate for the EFDCB protocol than FDCB since any connection interrupts should be minimized. The connection recovery time metric supplies evidence to support whether the EFDCB protocol can provide local fault-tolerance; thereby, offering better response in the face of connection failures. In other words, if the EFDCB protocol is more efficient at handling connection failures, it will provide lower connection recovery time than FDCB.

Testing focused on the amount of sustainable flow bandwidth will illustrate the developed protocol's ability to maintain flow demands given network failures over the global rerouting alternative. If EFDCB is only allowed to perform reconnections locally within the cluster, then in cases where the cluster cannot support the failed flow due to available resource limitations EFDCB will have to determine which flows to support and which to terminate (or drop). By looking at EFDCB’s ability to maintain flow demands while removing its ability to reroute from the source, insight about EFDCB’s efficiency is obtained. Data collected on the number of dropped packets and throughput is critical in determining the routing protocol’s ability to provide the primary task of QoS.

4.5 Parameters

4.5.1 System

Due to the distributed nature of the EFDCB, it is resilient to many of the parameters which affect its predecessor. The EFDCB fault-tolerant protocol either has the resources necessary to repair the failed connection or it does not. For this reason, the set of resource values (in this case bandwidth) under consideration when calculating the feasible paths is a parameter which affects the EFDCB. Also, the cluster topology is a parameter since this set of edges must be considered when calculating the feasible paths. When considering the FDCB protocol, many more parameters affect the system since this particular failure handling algorithm is centralized. One is the distance from the source to the destination. Referring back to the discussion of the failure cases in Section 3.3.2.2, for cases 1 and 2, using FDCB (the case where *n*5 is defunct in Figure 3.5) the message notifying the source of the failed connection has to make its way from the cluster-head (*n*8) to the source (*n*0). Further, during route negotiation messages must travel from the source to each cluster of which resources are required. This communication overhead contributes considerably to the overall performance of the FDCB system.

4.5.2 Workload

By manipulating node failure rate it is possible to illustrate the FDCB algorithm's inability to efficiently operate through failures.

The primary intent is to alter the network by removing the supporting nodes (that is, the nodes supporting the QoS connections). Manipulating these supporting nodes has the effect of increasing network dynamics; providing proper stimuli for the experimentation. The increase in failed connections will demonstrate whether or not the associated algorithm provides efficient protection in such challenging situations. Manipulating parameter has a major effect on the overall network bandwidth, queuing delay, mean path propagation delay, number of packets in the network, mean packet arrival rate, as well as other network parameters; however, the primary outputs of interested in here are connection recovery time, number of dropped packets, throughput and amount of sustainable flow bandwidth.

4.6 Failure Rate

Consider a large QoS supporting MANET in which ¾ of the nodes are filling a supporting role – that is ¾ of the nodes have no data to send but provide connectivity between the other source-destination pairs – while the other ¼ transmit/receive data. If the supporting nodes are gradually removed from the network, the number of possible connections to send data decreases. Further, since each intermediate node has a particular set of QoS capabilities, removing one node could prevent a source from transmitting its data (e.g., the removed node was the only node which could support the QoS requirement). In this experimentation, nodes are randomly removed and then, after 200 milliseconds, returned to the network once again. With this method, there is no concern for running out of network resources as long as the rate at which nodes return is greater than or equal to the rate nodes are removed (or the simulation is run for a sufficiently short period of time). This allows an experiment to run for any desired length of time. The MANET node failure rate is studied at 11 levels. The exact levels for this factor are shown in Table 4.1.

Table 4.1. Experimental Design for the First Phase of Experiments

Routing Algorithm	Experiment	Failure Frequency (ms)	Failures/Sec	Number of Flows	Average Cluster Bandwidth (Mbps)	Average Bandwidth Demand/Flow (Mbps)	Number of Runs
FDCB (or EFDCB)	1	40	25	6	400	100	10
	2	50	20	6	400	100	10
	3	60	16.70	6	400	100	10
	4	70	14.29	6	400	100	10
	5	80	12.5	6	400	100	10
	6	90	11.11	6	400	100	10
	7	100	10	6	400	100	10
	8	150	6.67	6	400	100	10
	9	200	5	6	400	100	10
	10	250	4	6	400	100	10
	11	300	3.33	6	400	100	10

Table 4.2 illustrates the experimental design aimed at sustained flow bandwidth response. For this set of experiments the network is initialized such that the available network bandwidth is extremely small. Also, EFDCB is not allowed to reroute from the source. In other words, the desire is to saturate the network so that EFDCB will quickly run out of resources and be forced to drop flows. With this idea in mind, all cluster links are reduced to half the original bandwidth. Also, the average bandwidth per flow is increased by increasing the number of source destination pairs as well as the demand for this additional traffic. Note that the average cluster bandwidth is calculated as the average bandwidth available from the incoming gateway node to the outgoing gateway node. Also note that FDCB is run under the same load and topology characteristics as just described.

Table 4.2. Experimental Design for the Second Phase of Experiments

Routing Algorithm	Experiment	Failure Frequency (ms)	Failures/Sec	Number of Flows	Average Cluster Bandwidth (Mbps)	Average Bandwidth Demand/Flow (Mbps)	Number of Runs
FDCB (or EFDCB)	12	40	25	12	200	125	10
	13	50	20	12	200	125	10
	14	60	16.70	12	200	125	10
	15	70	14.29	12	200	125	10
	16	80	12.5	12	200	125	10
	17	90	11.11	12	200	125	10
	18	100	10	12	200	125	10
	19	150	6.67	12	200	125	10
	20	200	5	12	200	125	10
	21	250	4	12	200	125	10
	22	300	3.33	12	200	125	10

4.7 Evaluation Technique

The evaluation technique used in this analysis is simulation. This technique affords the most commonsense approach to test the proposed hypothesis and therefore achieve the stated goal of showing that the developed MANET QoS routing algorithm is fault-tolerant. Moreover, since the source code of the simulation tool to be used (ns2 [18]) is readily available, this setting is the most likely environment to facilitate successful implementation of the desired EFDCB (and FDCB) functionality.

Using ns2, the system is configured as shown in Figure 4.2. Each node has a simulated *best-effort* omni-directional interface (used for cluster maintenance purposes) as well as a QoS supporting directional interface. At simulation start, all QoS links have the ability to support any one of the requested QoS connections; however, once a QoS connection has been established the associated intermediate nodes may or may not have the bandwidth available to support additional QoS requests. The arrows indicate gateway node and potential gateway node connections. Note that the clusters are predefined upon simulation initialization.

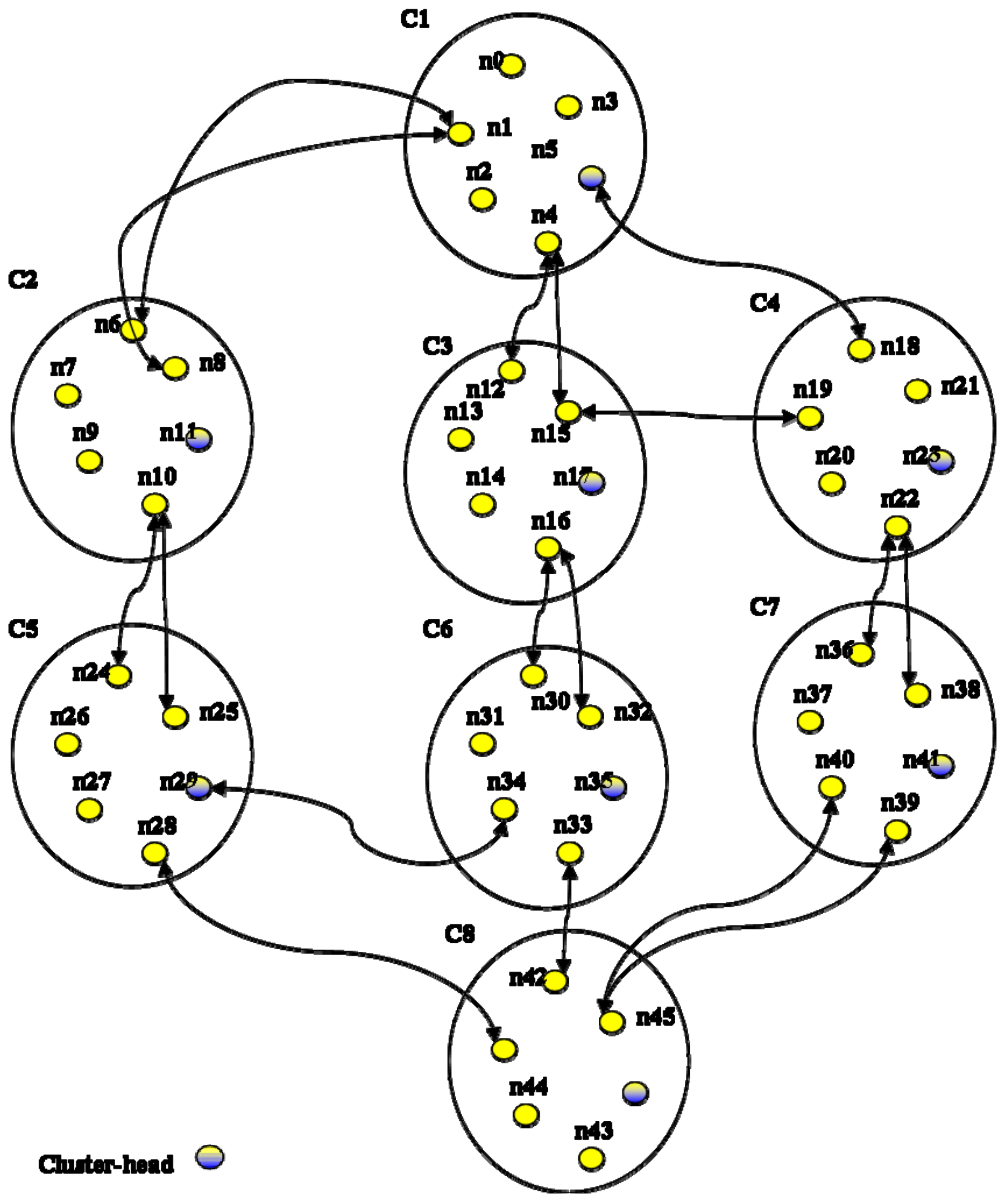


Figure 4.2. Experimental Network Architecture

Once a source-destination connection has been established, and the source begins to transmit the data, an intermediate node is randomly removed. This forces the routing algorithm to either re-route the traffic from the source (FDCB) or attempt to re-establish the connection (EFDCB) in the cluster associated with the removed node. In the case where the source and destination are separated only by a single node, it is likely that the local connection re-establishment option will be just as costly (in terms of recovery time) as having the source recalculate a route. As more nodes and clusters are added between the source and destination, the local algorithm will prevail in terms of time necessary to re-establish the path.

4.8 Experimental Design

Illustrating that the EFDCB algorithm is more fault-tolerant than the FDCB algorithm does not require a large number of complex experiments. In fact, a few well-formed experiments can achieve this task. The key is to demonstrate that the EFDCB algorithm is more expeditious at connection re-establishment. By repeatedly removing random nodes from the set of nodes supporting the source-destination connections (and periodically adding the removed nodes back to the network) the connection re-establishment procedure is initiated and the connection failure handling of the protocol under test is exercised. The experimental design for this study is illustrated in Table 4.1 and Table 4.2. Figure 4.2 illustrates the topology configuration of the experiments. All experiments are performed on both FDCB and EFDCB.

4.9 Preliminary Testing

Initial testing on the effects of distance between source-destination pairs for a constant failure rate proves that this is a main effect on performance for the FDCB algorithm – as expected.

Table 4.3. Initial Experimentation Based on Src-Dst Hop Distance

Experiment	Number of Hops	Number of Runs
1	3	10
2	4	10
3	5	10
4	6	10
5	7	10
6	8	10
7	9	10

These beginning tests show that the EFDCB system is minimally affected by this parameter. Figure 4.3 illustrates the results of these early tests. FDCB appears to display a significant growth in mean recovery time per failure response as hop counts increase. Experiments investigating effects of this parameter on EFDCB and FDCB were not explored further due to the costly process of topology generation; however, these opening experiments help to confirm the intuition that distance has a significant effect on the global nature of FDCB.

Additional preliminary testing was done to evaluate the failure rate at which the competing algorithms begin to be unsuccessful – assuming instantaneous failure discovery for both algorithms. The threshold failure rate at which FDCB begins to break down is approximately 25 failures per second (1 node failure every 40 milliseconds).

EFDCB continues to function properly up to 100 failures per second (1 node failure every 10 milliseconds). The limiting factor in this line of testing is the maximum recovery time. That is, if the length of time required to repair a broken link is longer than the interval between new broken links, the algorithm does not perform updates fast enough to have an accurate view of the required network state. This is similar to the concept of combinatorial stability. For FDCB this maximum recovery time is dependent upon the network size. For EFDCB the maximum recovery time is dependent upon the cluster size – a fraction of the total network. The main point here is that in both cases maximum recovery time is a consequence of network topology.

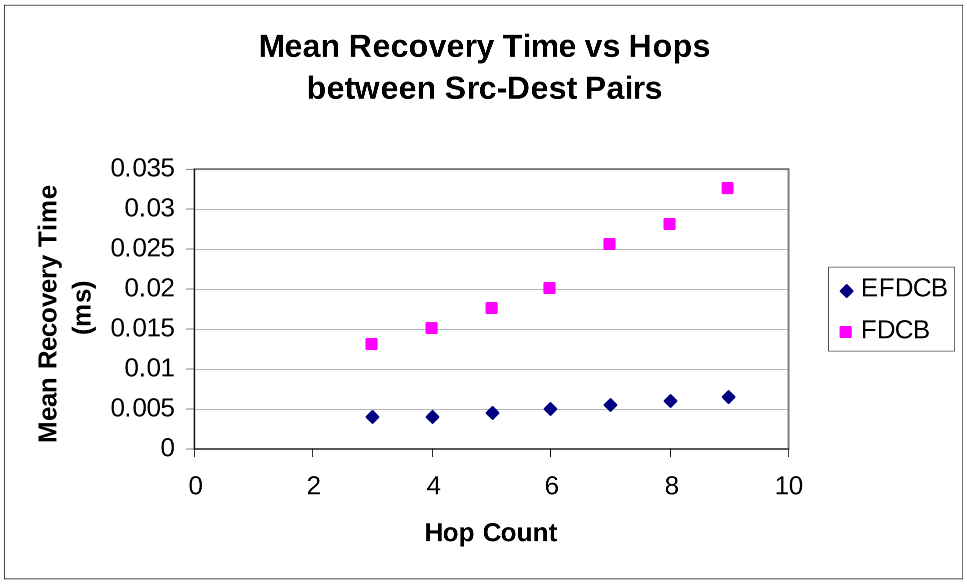


Figure 4.3. Recovery Time versus Number of Hops

4.10 Results of Simulations

Figure 4.4 illustrates the mean number of failures for each failure rate for each algorithm. In this graph it can be seen that 9 out of 11 times the EFDCB algorithm encountered more average failures than FDCB. Therefore the FDCB algorithm has a slight advantage in terms of offered load. That is, the EFDCB algorithm on average must handle more network failures than its predecessor during this experimentation. This is completely a product of the randomness of the failures.

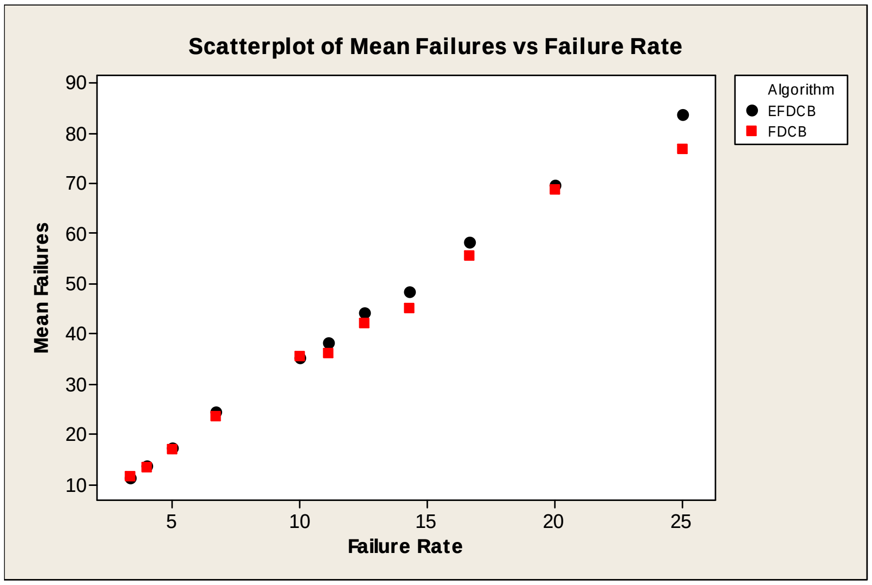


Figure 4.4. Mean Failures versus Failure Rate

The graph below (Figure 4.5) shows a raw data plot of recovery time versus failure rate for the failure rate values shown in Table 4.1. Recovery time is calculated as the sum of the individual recovery times (as defined in Section 3.6) of each reestablished connection for a given single experiment. The graph shows that both algorithms appear to have linear responses to linear increases in the failure rate. The data further demonstrates a spreading trend for FDCB as the rate of failures increase. This suggests a linear positive correlation where the variation of recovery time depends on the rate of failures. Also, note that in general much more variation in recovery time is recorded for

FDCB than for EFDCB. This makes sense since recovery time for FDCB is also affected by the number of hops between the cluster-head managing the failed node and the sources employing that failed node. That is, the farther this distance (in terms of hops), the more packets must be sent for route negotiation with intermediate clusters and more transmitted packets means more processing and propagation time.

The scatterplot of mean recovery time versus failure rate (Figure 4.6) demonstrates an obvious linear relationship between the two variables. Average recovery time is calculated as the average time spent handling failures per experiment (i.e., for the 1 node failure per 40ms experiment, run 10 times, the average recovery time is calculated, for the 1 node failure per 50ms experiment, run 10 times, the average recovery time is calculated and so on). Note that for every failure rate tested, EFDCB has a faster recovery time than FDCB by more than a factor of two.

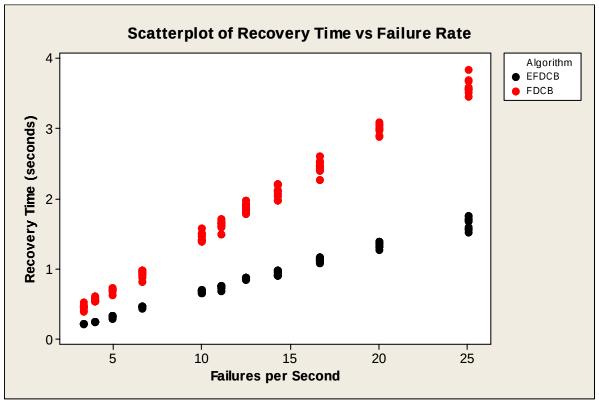


Figure 4.5. Raw Data Plot of Recovery Time versus Failure Rate

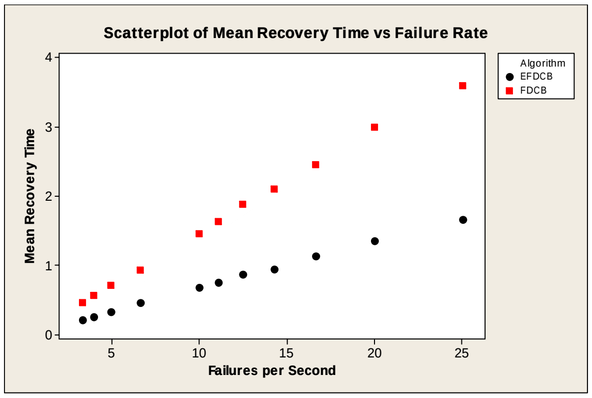


Figure 4.6. Mean Recovery Time versus Failure Rate

The mean recovery time fitted line plots for FDCB and EFDCB are shown in Figure 4.7 and Figure 4.8 respectively. The 80% prediction intervals capture the mean values indicating that these mean recovery time models fit the data well. The prediction interval provides a range within which one can expect the predicted response for a single sample to fall. Note that FDCB has more than twice the slope of EFDCB and hence more than twice the rate of increase for mean recovery time as the failure rate increases.

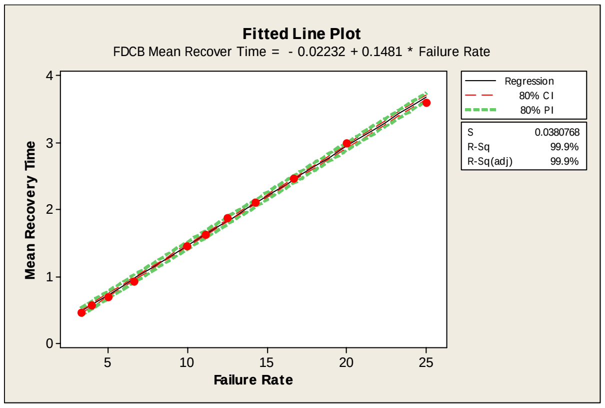


Figure 4.7. Fitted Line Plot for FDCB Mean Recovery Time versus Failure Rate

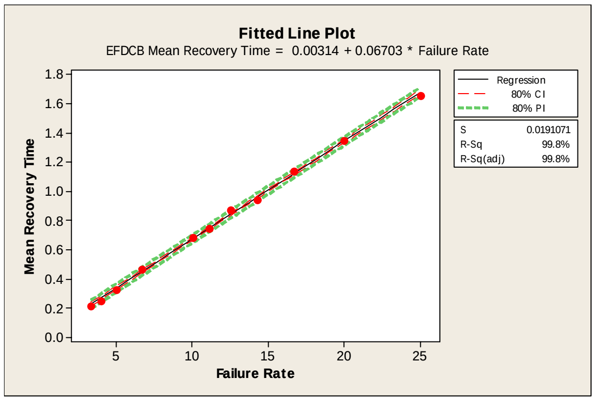


Figure 4.8. Fitted Line Plot for EFDCB Mean Recovery Time versus Failure Rate

The raw data illustrating the effects of failure rate on number of dropped packets is shown in Figure 4.9. The trend seen here is similar to that noted for recovery time; however, the relationship between the two variables appears to be less linear.

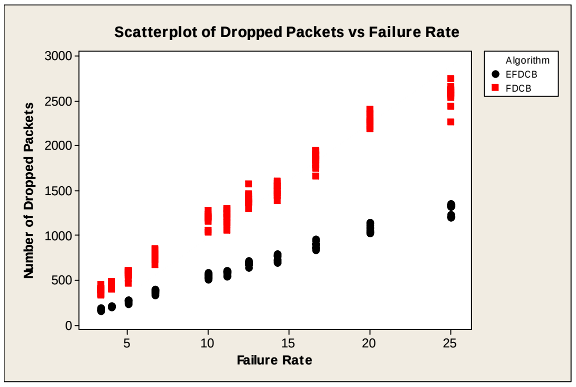


Figure 4.9. Raw Data Plot of Dropped Packets versus Failure Rate

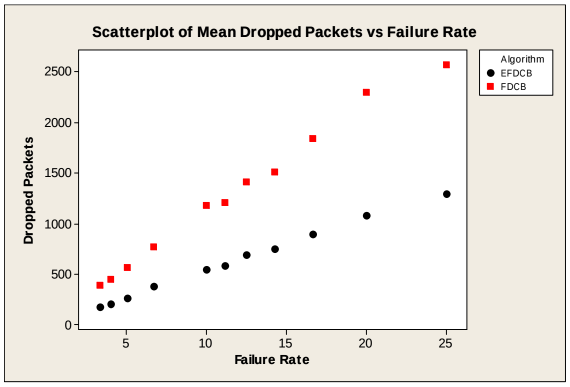


Figure 4.10. Mean Dropped Packets versus Failure Rate

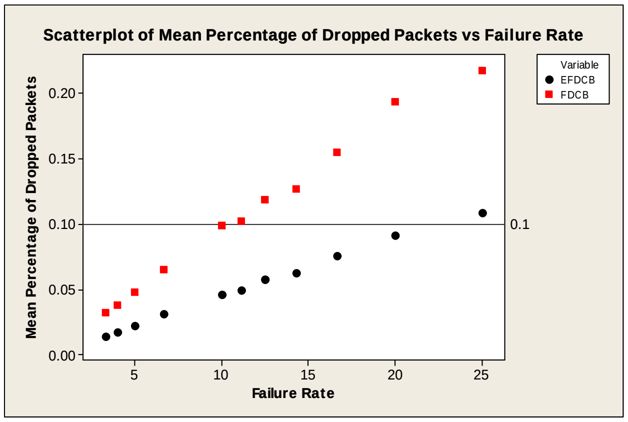
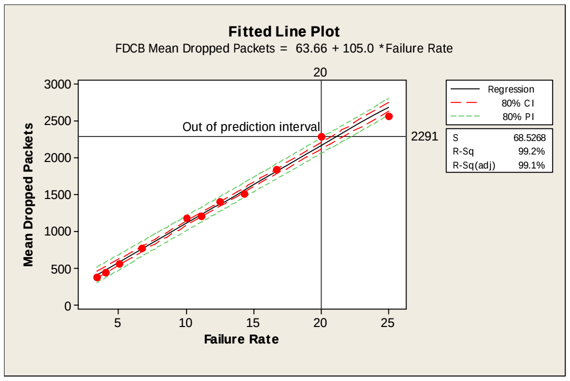


Figure 4.11. Mean Percentage of Dropped Packets versus Failure Rate

The scatterplot of mean dropped packets versus failure rate (Figure 4.10) as well as the fitted line plot (Figure 4.12) provide additional evidence to support the notion of inconsistency in true linearity of dropped packet response to failure rate input for FDCB. The fitted line plots shows a much wider prediction interval is necessary to capture the range within which one can expect the predicted response for a single sample to fall for

FDCB compared to EFDCB with 80% confidence. The difference in prediction interval ranges is even more noticeable if one notes that the maximum value for the y axis of the FDCB fitted line plot is more than twice that of the EFDCB fitted line plot. Further, Figure 4.12 shows that the prediction interval fails to capture one particular mean dropped packet value for FDCB. This deviation from consistent linearity for FDCB is most likely due to interaction caused by variations in the distance between the cluster-head where the failed node is located and any source directly impacted by the failed node. That is, the greater the distance (number of hops) between these two nodes, the more likely QoS packets will be dropped. This is because as distance increases the time required for the source to realize the node in its QoS path has failed also increases. On the other hand, EFDCB does give the impression of strong linear dropped packet response to linear increases in failure rate. This makes sense since EFDCB is somewhat resistant to variations in distance between the cluster-head associated with the failed node and the sources using resources on that failed node. Similar to the mean recovery time results, for every failure rate tested, EFDCB has less mean dropped packets than FDCB by more than a factor of two.



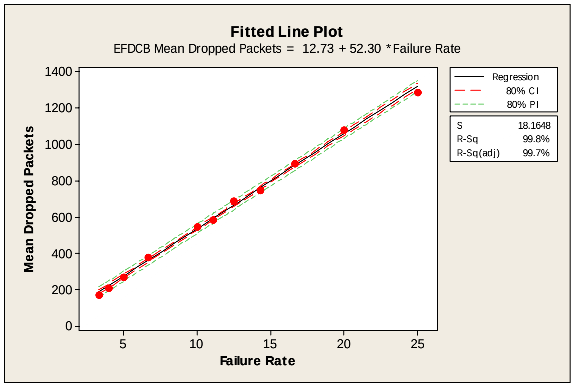


Figure 4.12. Fitted Line Plot for FDCB Mean Dropped Packets vs Failure Rate

Figure 4.13. Fitted Line Plot for EFDCB Mean Dropped Packets vs Failure Rate

The throughput versus failure rate results (Figure 4.14 and Figure 4.15) show that both algorithms appear to have a general linear response. Again the deviations from concise linearity can be attributed to the effects of distance interacting with the failure rate. Figure 4.16 shows the percentage of optimal throughput versus failure rate. Percentage of optimal throughput is calculated as:

$$Percentage\,of\;optimal\,throughput = \frac{realized\;throughput}{throughput\;without\;\,failures}\,.$$

The mean throughput for EFDCB never falls below the 90% threshold; however, FDCB dips down to 79% of the optimal throughput.

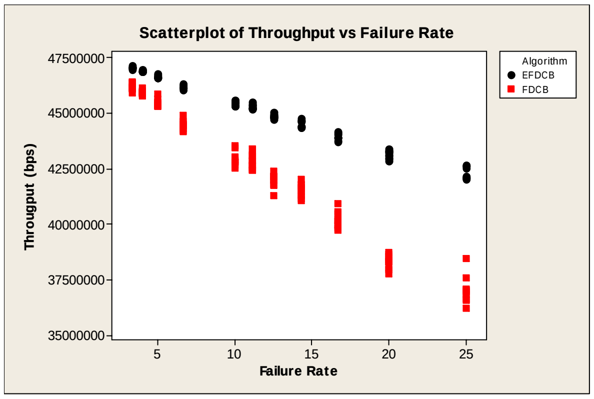


Figure 4.14. Raw Data plot of Throughput versus Failure

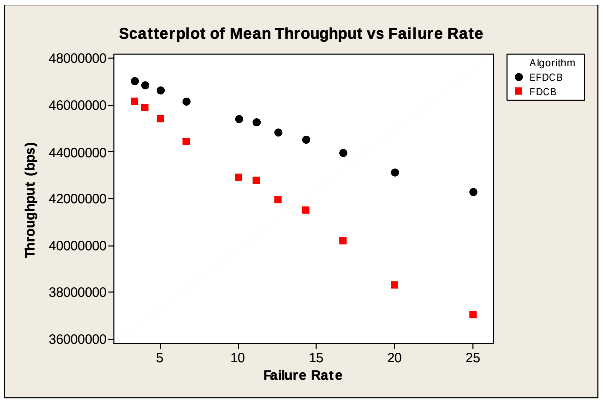


Figure 4.15. Mean Throughput versus Failure Rate

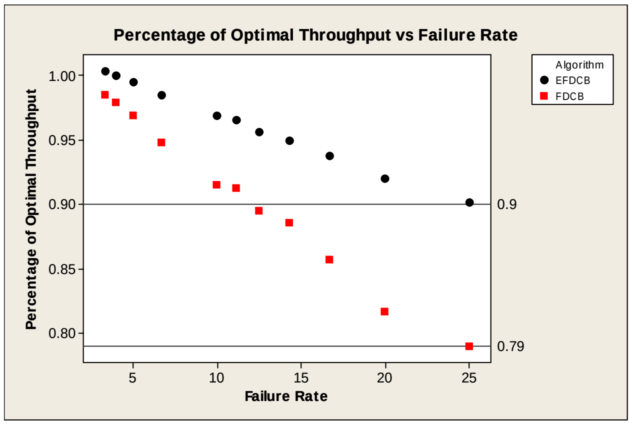


Figure 4.16. Percentage of Optimal Throughput versus Failure Rate

The bar chart of sustained flow bandwidth versus failure rate (Figure 4.17) shows that with a saturated network the "pure" local protocol (EFDCB with rerouting functionality removed in this case) is able to compete effectively with the global rerouting algorithm for failure rates up to 6.7 failures per second. Beyond 6.7 failures per second, the global rerouting algorithm (FDCB) is significantly better at finding new routes when link failures occur. The EFDCB protocol is designed to invoke global recovery in cases where the flow is no longer sustainable by local resources. The bottom-line is that EFDCB obtains the advantages of both the "pure" global and the "pure" local recovery methods.

It is interesting to note that for 16.7 node failures per second and above, sustained flow bandwidth remains constant. This is due to the criteria used to pick the random node for failure. The algorithm used to fail nodes first checks to see if the failure of the node will render the flow irreparable. That is, if removal of a cluster node makes connectivity through the cluster impossible, no node will be removed in the cluster; therefore, the upper bound on the worst case mean sustained flow bandwidth is less than or equal to the sum of available bandwidth through the clusters – without breaking connectivity through the clusters. It is worth noting that the method implemented to arbitrate between the set of flows to support and the set of flows to drop involves picking the set that optimizes the sustained flow bandwidth.

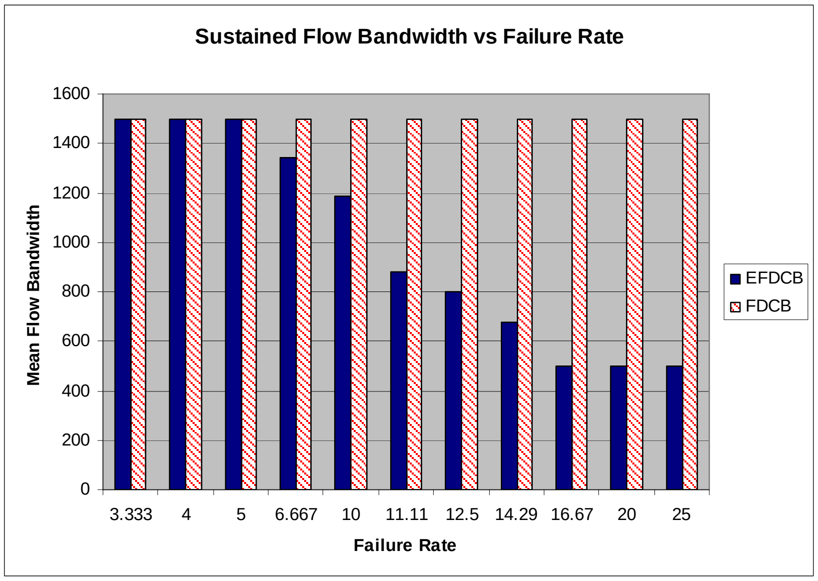


Figure 4.17. Sustained Flow Bandwidth versus Failure Rate

4.11 Summary

Preliminary testing revealed that distance between source-destination pairs is a primary driver on the recovery time response for FDCB. More precisely, the distance between the cluster-head of the failed node and sources employing resources on the failed node has a strong effect on recovery time response for FDCB. This was an expected result; however, demonstrating this in simulation aided in reasoning about irregularities in predicted responses for FDCB. That is, difficulties in generating a sufficiently accurate linear model (80% confidence) for dropped packets and throughput for FDCB are attributed to the interaction of distance with failure rate. EFDCB is much more predictable since this interaction does not present a problem for this local algorithm.

The recovery time response for increasing failure rates showed that EFDCB is predictably twice as fast as FDCB for every failure rate tested. The dropped packet response to increasing failure rates showed that EFDCB suffers from less dropped packets than FDCB by more than a factor of two. This makes sense since recovery time showed a similar result. The throughput response to increased failure rate demonstrates that EFDCB drops to just above 90% of the optimal throughput at 25 failures per second. FDCB goes down to 79% of the optimal throughput at this same failure rate.

Tests investigating the effect of failure rate on sustained flow bandwidth reveal that a pure local algorithm is less effective than FDCB for failure rates equal to or greater than 6.7 failures per second. Although the local algorithm performs a local optimization on the sustainable flow bandwidth when adjudicating what flows to support, since FDCB has the ability to reroute the traffic from the source it can find routes not available to this pure local procedure. The take away from this line of testing is that although a local fault-tolerant algorithm provides significant recovery and QoS disruption time improvement, to be effective in all cases, it must be coupled with an efficient rerouting alternative for situations where required local resources are not available. EFDCB incorporates this functionality.

For an example of how the captured metrics relate to QoS, consider the maximum dropped packet rate tolerable for VoIP traffic. First note that this value is dependent upon the Coder/Decoder (codec) used. RFC3714 states that "voice quality begins to deteriorate for many codecs around a 10% drop rate". Figure 4.11 shows that for FDCB this occurs around 10 failures per second. For EFDCB this happens around 22 failures per second. This clearly shows EFDCB is more likely to provide the required QoS for this particular application given the challenged network tested here.

The experimental design presented in this chapter described 22 experiments aimed at illustrating the EFDCB protocol's ability to handle fault-tolerance. The interpreted results showed that EFDCB excels over FDCB at this challenge which is the goal of this work. The next chapter uses these results to highlight the conclusions and future work.

V. Conclusions and Recommendations

5.1 Chapter Overview

This chapter discusses and concludes the implications of this thesis. The conclusions focus on the benefits of local fault-tolerance in QoS supporting mobile wireless networks. Following the conclusions, future research items are discussed.

5.2 Conclusions of Research

This work developed a distributed fault-tolerant routing protocol for QoS supporting hybrid mobile ad hoc networks with the aim of mitigating QoS disruption time when network failures occur. In Chapter One, the problem considered was introduced along with some background and the focus of this research. Chapter Two presented introductory material on the QoS problem as it applies to wired networks and built on this with the topic of QoS in the MANET. Chapter Two also covered an overview of literature that supports key design decisions made in development of the distributed fault-tolerant cluster based QoS protocol. Different solution methods that exist in the literature were discussed as well. In Chapter Three, the extended-Fully Distributed Cluster Based protocol was demonstrated and described. In Chapter Four, an experimental design was developed, the collected data was analyzed and conclusions based on the discovered trends were annotated.

This thesis demonstrates that the traditional method of rerouting QoS traffic from the source given a link failure yields serious negative QoS disruption consequences; however, an efficient local fault-tolerant algorithm can significantly mitigate the time required to reestablish the connection. The ultimate advantage gained by mitigating the time to reestablish connections is a reduction in QoS disruption.

The preliminary testing results from Chapter Four show that a pure rerouting algorithm exhibits significant growth in recovery time as the distance between the source and destination increases. This makes sense when one considers the process of message exchanges that must occur for a source to reroute its traffic. More specifically, once the cluster-head associated with the failed node realizes a node has failed it sends a 'failed node' message to any source which was using resources on the failed node. Hence, the elapsed time between the moment the node fails to the time the source is notified equal to the time required for the cluster-head to notice the failure plus the time for the failed node message to propagate to the source. At this point, the source must determine a new feasible route. Upon determining a new route, the source must negotiate its demands with the cluster-heads of all clusters that have resources the sources desires to use and then wait for responses from these cluster-heads. It's easy to see that an increase in the number of clusters between the source and destination will have a significant impact on the time required to complete the necessary negotiations. The recovery time results showed that EFDCB is more than two times faster than the global rerouting alternative for all failure rates tested. The dropped packet and throughput results reflected similar outcomes.

5.3 Recommendations for Future Research

This early work is part of a bigger effort to enable robust H-MANET functionality, the critical extension required to empower the GIG with the capabilities envisioned by current civilian and military leadership, as discussed in Chapter One. EFDCB possess the ability to cluster mobile nodes, route QoS traffic, and handle link failures. For the collection of experiments performed here, clustering and routing were bootstrapped. The focus was on EFDCB's ability to handle link failures. Having demonstrated efficient fault-tolerance, the next logical step in experimentation of EFDCB is the clustering processes. This makes sense since the clustered architecture is the fundamental structure on which all other portions of the protocol are built. That is, the failure handling, as well as the routing protocol, requires this clustered wireless network architecture in order to be functional. With the clustering protocol implemented, much more comprehensive mobility testing (e.g., effects of cluster size on fault-tolerance and routing) can be performed. This mobility testing is vital if it is to be shown that EFDCB makes a significant contribution to the overall hybrid mobile wireless network effort.

Bibliography

[1] Al-Karaki, J.N., A.E. Kamal, and R. Ul-Mustafa. "On the optimal clustering in mobile ad hoc networks," *Proceedings of the IEEE Consumer Communications and Networking Conference*. 71-76. New York: First IEEE,2004.

[2] Alberts, D.S., Garstka, John J., and Stein, Frederick P. *Network Centric Warfare: Developing and Leveraging Information Superiority*. (2nd Edition). CCRP, 1999

[3] Applegate, D. and M. Thorup. "Load optimal MPLS routing with N + M labels," *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.* 555-565. IEEE. 2003.

[4] Broch, J., D.B. Johnson, and D.A. Maltz, *The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks*, in *Internet Draft*, IETF, Editor. 1998.

[5] CDRUSSTRATCOM, *Joint Concept of Operations for Global Information Grid NetOps*, DoD, Editor. 2005. p. 65.

[6] Cen, S., et al. "A Distributed Real-Time MPEG Video Audio Player," *Proceedings of the Fifth International Workshop on Network and Operating System Support of Digital Audio and Video* (*NOSSDAV '95*). 151-162. London, UK: Springer-Verlag, 1995.

[7] Chakrabarti, S. and A. Mishra, *QoS issues in ad hoc wireless networks.*, in *Communications Magazine, IEEE*. 2001. p. 142-148.

[8] Chen, S., *Routing Support For Providing Guaranteed End-to-end Quality-of-Service*, PhD dissertation. University of Illinois: Urbana. 1999.

[9] Cormen, T.H., et al. *Introduction to Algorithms*. (2nd Edition). Cambridge, MA:McGraw-Hill Higher Education, 2003

[10] Department of Defense, *Data Sharing in a Net-Centric Department of Defense (DoD)*. DoD Directive 8320.2. Washington: GPO, December 2, 2004.

[11] Department of Defense, *Use of Commercial Wireless Devices, Services, and Technologies in the Department of Defense (DoD) Global Information Grid (GIG)*. DOD Directive 8100.2. Washington: GPO, April 14, 2004.

[12] Dimitriadis, G. and F.N. Pavlidou. "Clustered fisheye state routing for ad hoc wireless networks," *Proceedings of the Mobile and Wireless Communications Network, 2002. 4th International Workshop*. 207-211. IEEE, 2002.

[13] Fleischer, L.K. "Approximating fractional multicommodity flow independent of the number of commodities," *Proceedings of the Foundations of Computer Science, 1999. 40th Annual Symposium*. 24-31. IEEE, 1999.

[14] Ghosh, R. and S. Basagni, "Limiting the impact of mobility on ad hoc clustering," *Proceedings of the 2nd ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*. 197-204, ACM Press: 2005.

[15] Goktas, F., J.M. Smith, and R. Bajcsy. "Telerobotics over communication networks," *Proceedings of the 36th IEEE Conference*. 2399-2404. IEEE, 1997.

[16] Guangyu, P., M. Gerla, and C. Tsu-Wei. "Fisheye state routing in mobile ad hoc networks," *Proceedings of the ICDCS Workshop on Wireless Networks and Mobile Computing*. 71-78, ICDCS 2000.

[17] Gupta, P. and P.R. Kumar. "The capacity of wireless networks," *Information Theory, IEEE Transactions*, 46(2): 388-404 (March 2000).

[18] *The network simulator--ns-2.* Version 2.29, Computer Software. Informaiton Sciences Institute, Marina del Rey CA, 2006.

[19] Larsen, K.S., *Dictionary of Computer Science, Engineering and Technology*, in *Dictionary of Computer Science, Engineering and Technology*, CRC Press LLC, 2001.

[20] Mellouk, A. "Quality of Service Dynamic Routing Schemes for Real Time Systems in IP Network," *Proceedings of the Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*. 93-93. *ICN/ICONS/MCL*, 2006.

[21] Mitra, D. and K.G. Ramakrishnan. "A case study of multiservice, multipriority traffic engineering design for data networks," *Proceedings in the Global Telecommunications Conference* (*GLOBECOM '99*). 1077-1083. GLOBECOM, 1999.

[22] Nargunam, A.S. and M.P. Sebastian. "Fully distributed cluster based routing architecture for mobile ad hoc networks," *Proceedings in the Wireless And Mobile Computing, Networking And Communications* (*WiMob'2005*), *IEEE International Conference*. 383-389. IEEE, 2005.

[23] Pearlman, M.R. and Z.J. Haas. "Determining the optimal configuration for the zone routing protocol," *Selected Areas in Communications, IEEE Journal*, 17(8): 1395-1414 (August 1999).

[24] Penkoske, J. "DISA: DoD's Employer of Choice." Excerpt from unpublished article. Military Information Technology Online Archives pag. http://www.military-information-technology.com/article.cfm?DocID=1423. 20 May 2006.

[25] Puri, A. and S. Tripakis, *Algorithms for Routing with Multiple Constraints*. *Technical Report*, 2001. EECS Department, University of California, Berkeley. 2001 (Report Number UCB/ERL M01/7).

[26] Shigang, C. and K. Nahrstedt. "Distributed quality-of-service routing in ad hoc networks," *Selected Areas in Communications, IEEE Journal*, 17(8): 1488-1505 (August 1999).

[27] Shigang, C. and K. Nahrstedt. "On finding multi-constrained paths," *Proceedings in the Record 1998 IEEE International Conference*. 874-879. IEEE, 1998.

[28] Tran, N. and K. Nahrstedt. "Active adaptation by program delegation in video on demand," *Proceedings in the IEEE International Conference.* 96-105. IEEE, 1998.

[29] Widyono, R., *The Design and Evaluation of Routing Algorithms for Real-time Channels*. *ICSI Technical Report, June 1994*. University of California at Berkeley & ICSI, June 1994 (Report Number tr-94-024).

[30] Yuan, X.. "Heuristic algorithms for multiconstrained quality-of-service routing," *Networking, IEEE/ACM Transactions*, 10(2): 244-256 (April 2002).

Vita

Captain Larry C. Llewellyn II graduated from South Point School in Belmont, North Carolina. He entered undergraduate studies at the University of North Carolina at Charlotte, in Charlotte, North Carolina, where he graduated with a Bachelor of Science degree in Electrical Engineering Technology and received his commission in December 2001. His first assignment was at the Defense Information Systems Agency as a Global Network Operations Watch Officer. His next assignment is at the Air Force Communications Agency at Scott AFB, Illinois.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)* 22-03-2007	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* May 2005 – March 2007
4. TITLE AND SUBTITLE Distributed Fault-tolerant Quality Of Service Routing in Hybrid Directional Wireless Networks	**5a. CONTRACT NUMBER**	
	5b. GRANT NUMBER	
	5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Llewellyn II, Larry C., Captain, USAF	**5d. PROJECT NUMBER**	
	5e. TASK NUMBER	
	5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology, Graduate School of Engineering and Management (AFIT/EN), 2950 Hobson Way, Building 640, WPAFB OH 45433-8865	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GE/ENG/07-15	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Office of Scientific Research, Dr. David Luginbuhl, 875 N. Randolph Street, Arlington, VA 22203-1768, (703)696-6207, david.luginbuhl@afosr.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)**	
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)	

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

fault-tolerant, quality-of-service, routing, network flows, distributed

14. ABSTRACT

This thesis presents a distributed fault-tolerant routing protocol (EFDCB) for QoS supporting hybrid mobile ad hoc networks with the aim of mitigating QoS disruption time when network failures occur. The experimental design presented in this thesis describes 22 experiments aimed at illustrating EFDCB's ability to handle fault-tolerance. The interpreted results show that EFDCB excels over a global rerouting protocol at this challenge which is the goal of this work.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr Kenneth M. Hopkinson
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	130	(937) 255-6565, ext 4579 (Kenneth.hopkinson@afit.edu)